# TI - 99/4A

# TI EXTENDED BASIC

## MANUALE IN ITALIANO

# TI EXTENDED BASIC

manuale d'uso in italiano
traduzione a cura di

**CLAUDIO DE NISI**

per la **DENCAM'S** software

distribuzione **NOV.EL.'81**

fotocopiato in proprio - Roma, giugno MCMLXXXV

Il fascino dei linguaggi...
la poesia degli umani,
il mistero delle macchine,
il progresso che imposta
l'attitudine mentale.

*Claudio De Nisi*

Si ringrazia:

| | | |
|---|---|---|
| la | **TEXAS INSTRUMENTS** | per la realizzazione del TI-99/4A, sue periferiche ed in particolare del package TI-WRITER Word Processor utilizzato per la stesura di questa traduzione italiana. |
| la | **EPSON** | per la stampante RX-80F/T utilizzata per la scrittura. |
| la | **NOV.EL.'81** | per averne permesso la realizzazione e distribuzione. |

ed in particolare:

FRANCO BARTOCCINI
NICOLA CAMINITI
ENZO COLORITO
FAUSTO MINUZZO
ALFIERO VENTURA

# TI EXTENDED BASIC

==========================================================================
==========================================================================
==========================================================================

PER L'HOME COMPUTER TI-99/4A

Il TI EXTENDED BASIC é un potente linguaggio di programmazione ad alto livello che aumenta le capacita' del computer TI-99/4A. Comprende le seguenti caratteristiche:

- Piu' di quaranta tra comandi, istruzioni, funzioni e sottoprogrammi nuovi o potenziati.
- Istruzioni multiple che garantiscono piu' velocita' ed efficienza.
- Possibilita' di comandare gli sprites per la grafica in movimento.
- Uso di sottoprogrammi nominali che vi danno la possibilita' di conservare su dischetto le routines piu' comunemente usate in modo da riutilizzarle quando necessarie.
- Capacita' di eseguire un programma richiamandolo da un altro.
- Controlli personalizzati degli errori, degli avvisi di errore e dei punti di arresto nei programmi.
- Controllo diretto sullo schermo dell'immissione dei dati e dei risultati.
- Possibilita' di sviluppare ed eseguire programmi in lingaggio Assembly TMS9900 se e' collegata l'unita' di espansione della memoria.

# INDICE

APPENDICI

# CAPITOLO 1

## INTRODUZIONE

# INTRODUZIONE

**CARATTERISTICHE**

L'Extended BASIC della Texas Instruments e' un potente linguaggio di programmazione per l'uso dell'Home Computer TI-99/4A. Ha le caratteristiche che ci si aspettano da un linguaggio ad alto livello ed altre generalmente non offerte da molti altri linguaggi, compresi quelli installati su grossi e costosi calcolatori.

Nella versione estesa sono presenti oltre 40 comandi, istruzioni, funzioni o sottoprogrammi nuovi o supplementari estesi. Il linguaggio TI BASIC Esteso va oltre il TI BASIC per potenziare la capacita' e la flessibita' d'impiego del vostro sistema con l'aggiunta delle seguenti caratteristiche.

- *INPUT ED OUTPUT* - (Ingresso ed uscita). La dichiarazione ACCEPT permette l'entrata dei dati da qualsiasi punto dello schermo. Si puo', usando questa istruzione, pulire lo schermo, accettare solo i caratteri voluti e dare un limite al numero dei caratteri da immettere. La dichiarazione DISPLAY e' stata potenziata per permettere di mostrare dati in qualsiasi punto dello schermo. Si sono anche aggiunte le dichiarazioni DISPLAY... USING, PRINT...USING, ed IMAGE per facilitare la disposizione dei dati sul video e le unita' periferiche.

- *SOTTOPROGRAMMI* - E' possibile redigere sottoprogrammi con variabili locali (che hanno cioe' significato solo nell'ambito di quel sottoprogramma). I sottoprogrammi di maggiore uso possono essere memorizzati su dischetti ed aggiunti ai programi secondo le necessita'. Le dichiarazioni incluse sono SUB, SUBEND e SUBEXIT (Sottoprogramma, Fine sottoprogramma ed Uscita sottoprogramma). Si e' aggiunto il comando MERGE (fusione) e si e' modificato il SAVE per permettere la fusione di programmi memorizzati su dischetti.

- *SPRITES* - Cosi' si definiscono gli speciali disegni grafici che si possono spostare sullo schermo (sprite significa folletto). Ai fini della funzione SPRITES sono stati inclusi i seguenti sottoprogrammi: COINC (Coincide), DELSPRITE (cancella sprite), DISTANCE, LOCATE (Localizza), MAGNIFY (Ingrandisci), MOTION (Movimento), PATTERN (Disegno), POSITION e SPRITE. COLOR e CHAR sono stati ampliati per cui possono anch'essi agire sugli sprites.

- *FUNZIONI* - Sono stati inclusi nel BASIC ESTESO :MAX, che restituisce il piu' grande fra due numeri, MIN, che restituisce il minore, e PI che restituisce il valore del pi greco.

- *MATRICI* - Le matrici possono essere dimensionate fino ad un massimo di sette anziche' di tre.

- *RIPETIZIONE STRINGA* - La funzione RPT$ permette la ripetizione di una stringa.

● *CONTROLLO DEGLI ERRORI* - E' possibile scegliere quale azione far eseguire al computer quando nel corso dell'elaborazione si presentano quelle condizioni di errore che in TI Basic causano un messaggio di avvertimento per un errore lieve, l'arresto del programma con messaggio per un errore grave o un "breakpoint" o punto d'arresto. Le nuove istruzioni che permettono questo sono: ON WARNING, ON ERROR, ON BREAK. RETURN e' stato modificato per un uso anche in questo caso. L'istruzione CALL ERR puo' essere usata per determinare la natura di un errore occorso nel programma.

● *RUN COME ISTRUZIONE* - RUN puo' essere usato come istruzione oltre che come comando. Inoltre RUN e' stato modificato per permettere di far partire un programma specifico. Come risultato da un programma se ne puo' lanciare un altro e soprattutto si possono scrivere programmi di qualsiasi dimensione spezzandolo in varie parti collegabili cosi' tra loro.

● *PARTENZA AUTOMATICA DEL PROGRAMMA* - Scegliendo l'opzione 2 del TI Basic Esteso il sistema cerca sul dischetto del drive 1 il programma LOAD; se esiste questo viene caricato in memoria ed eseguito.

● *ISTRUZIONI MULTIPLE* - Il TI BASIC Esteso consente che una linea contenga piu' di un'istruzione. Questo rende piu' veloce l'esecuzione del programma, risparmia memoria e permette che su di un'unica linea risieda un insieme logico (Ad esempio il ciclo FOR...NEXT).

● *IF-THEN-ELSE* - Queste istruzioni ora permettono le assegnazioni come conseguenza di una comparazione. Questo ampliamento permette l'uso di istruzioni come "IF X<4 THEN GOSUB 240 ELSE X=X+1".

● *ASSEGNAZIONI MULTIPLE* - E' possibile assegnare uno stesso valore a piu' variabili nell'istruzione LET, con risparmio di memoria e maggior efficienza.

● *COMMENTI* - Oltre all'istruzione REM, i commenti possono esere posti anche alla fine delle linee di programma, permettendo una dettagliata documentazione interna dei programmi.

● *SUPPORTO LINGUAGGIO ASSEMBLY* - Con la scheda di espansione di memoria (venduta separatamente), e' possibile immettere ed eseguire sottoprogrammi in linguaggio ASSEMBLY del TMS9900 mediante i sottoprogrammi INIT, LOAD, LINK e PEEK.

● *INFORMAZIONI* - Il comando SIZE e' stato aggiunto per poter conoscere in ogni momento la quantita' di memoria rimasta ancora libera. Il sottoprogramma VERSION da' un valore che indica la versione del BASIC che si sta utilizzando.Il sottoprogramma CHARPAT restituisce una stringa di caratteri che indica il pattern di definizione di un carattere.

● *ESPANSIONE DI MEMORIA* - Il TI Extended BASIC permette l'utilizzo dell'espansione di memoria esterna, che permette la scrittura di programmi piu' vasti.

## DIFFERENZE RISPETTO AL TI BASIC

I miglioramenti descritti precedentemente hanno creato alcuni piccoli cambiamenti in altre aree del TI BASIC. Per questo motivo, alcuni programmi scritti in TI-99/4 BASIC non possono essere eseguiti in TI Extended BASIC.

Il massimo spazio per il programma e' ora di 864 bytes piu' piccolo che in TI BASIC. Se avete l'espansione di memoria esterna potete pero' scrivere programmi piu' lunghi.

Non sono piu' disponibili i caratteri dei sets 15 e 16. Questa area di memoria e' usata dal TI Extended BASIC per registrare la traccia degli sprites.

Molti programmi scritti in TI BASIC possono essere anche eseguiti in TI Extended BASIC senza difficolta'. In alcuni casi tuttavia, come nell'uso di una parola riservata al TI Extended BASIC come variabile in un programma TI BASIC, i programmi scritti in TI BASIC non possono essere eseguiti in TI Extended BASIC. Tuttavia si possono sempre caricare programmi TI BASIC con l'Extended BASIC. Naturalmente i miglioramenti del TI Extended BASIC non potranno essere eseguiti correttamente in TI BASIC.

## COME USARE QUESTO MANUALE

Questo manuale presuppone che abbiate gia' esperienza di programmazione con il TI BASIC. Le istruzioni, i comandi e le funzioni che sono uguali al TI BASIC, saranno spiegate solo brevemente. Per una piu' dettagliata spiegazione, consultare il mauale d'uso del TI-99/4A.

Le caratteristiche del TI Extended BASIC sono invece spiegate nei dettagli ed illustrate con esempi e programmi. Per ottenere il massimo vantaggio leggere attentamente questo manuale, ricopiando ed eseguendo i programmi di esempio ed osservare come funzionano. Andrebbero inoltre riviste anche le caratteristiche che sono rimaste invariate dal TI BASIC. Potreste scoprire di trascurare un'istruzione utile, o un nuovo modo per usare le istruzioni in combinazioni diverse.

Il resto di questo capitolo illustra i fondamenti del TI Extended BASIC. Il secondo capitolo spiega le caratteristiche del TI Extended BASIC e comprende un esempio dettagliato dell'immisssione di un programma. Il terzo capitolo illustra il modo di operare del sistema TI Extended BASIC. Il quarto, e' una sezione di riferimento che illustra, in ordine alfabetico, tutte le istruzioni, i comandi e le funzioni del TI Extended BASIC.

L'appendice 14 contiene molte informazioni utili, compresi i codici dei caratteri ASCII, i codici di errore, della tastiera e le istruzioni per aggiungere i suffissi alle parole del sintetizzatore.

## COME USARE IL COMPUTER

Prima di usare il computer con il TI Extended BASIC, occorre inserire il modulo SSS (*Solid State Software*). Se il computer e' spento, inserirlo lentamente nell'apposita guida a destra della tastiera finche' non si blocca. Quindi accendere il computer. (Se sono collegate periferiche queste vanno accese prima del computer). Apparira' cosi' sullo schermo il titolo principale. Premere un tasto qualsiasi. Comparira' la lista di selezione. Il TI EXTENDED BASIC e' il secondo. Premere 2 per selezionarlo.

## OPERARE CON L'EXTENDED BASIC

Con l'Extended Basic si dispone di tre modi operativi principali: Modo Comando, Modo Edit e Modo Run.

Il Modo Comando e' quello nel quale ci si trova subito dopo aver selezionato 2 dalla lista principale. Potete inserire comandi e istruzioni eseguibili direttamente, e linee di programma.

Il Modo Edit serve per correggere linee esistenti di un programma. Per entrare in Modo Edit, digitare il numero di linea e premere i tasti **FCTN E** (UP) o **FCTN X** (DOWN) (in Extended basic non esiste il comando EDIT seguito da un numero di linea come in TI BASIC ). La linea specificata e' visulizzata sullo schermo. Potete cambiarla digitando una diversa linea, cambiarne o cancellarne una parte, o cambiare il numero utilizzando i tasti di correzione spiegati di seguito. Si entra in modo Edit anche premendo i tasti **FCTN 8** (REDO), per ripetere l'ultima cosa inserita (comando o linea di programma).

In Modo Run, viene eseguito il programma. Potete arrestare l'esecuzione del programma solo premendo **FCTN 4**, che causa un breakpoint (punto d'arresto), o **FCTN +** (QUIT). *Nota:* QUIT causa anche la cancellazione dell'intero programma, ed il ritorno alla maschera d'accensione, e possono essere anche cancellati dei dati all'interno di files rimasti aperti. Per lasciare l'Extended BASIC e' quindi consigliabile l'uso di BYE in luogo di QUIT (FCTN +).

**FCTN 2** (INSERT) abilita il computer ad accettare l'inserimento di caratteri da dove si trova il cursore in poi. Ogni successivo carattere che viene digitato e' inserito nella posizione del cursore, ed i caratteri alla destra di questo vengono fatti slittare di una posizione verso destra per ogni nuovo carattere premuto. L'inserimento continua finche' non viene premuto **ENTER** o un altro tasto funzione. Vengono perduti i caratteri che vengono via via a trovarsi oltre il limite massimo di lunghezza della linea.

**FCTN 1** (DELETE) elimina il carattere sul quale viene posizionato il cursore, e tutti i caratteri alla destra del cursore slittano verso sinistra finche' rimane premuto **FCTN 1**.

**FCTN 4** (CLEAR) esegue varie funzioni a seconda del modo nel quale si sta operando. Se vi trovate in modo Edit, vengono ignorati i cambiamenti eseguiti sulla linea, compreso **FCTN 3** (ERASE), e il computer ritorna in modo Comando. Se vi trovate in modo Run, il programma si ferma con un breakpoint. Se vi trovate in modo Comando, i caratteri che avete digitato nella linea vengono cancellati. Quando si usa **FCTN 4** per fermare un programma, continuare a tenere premuti i tasti finche' l'Extended BASIC non riconosce il breakpoint.

**FCTN =** (QUIT) fa tornare il computer alla maschera principale d'accensione. *Quando premete FCTN = (QUIT), tutti i dati e programmi vengono eliminati dalla memoria del computer. Se state utilizzando il sistema a dischi, alcuni dei vostri files potrebbero essere danneggiati.* Lasciare il TI Extended BASIC digitando BYE invece di QUIT.

**ENTER** indica che avete terminato di digitare le informazioni della linea e fa in modo che esse siano trattate dal computer.

# CAPITOLO 2

## PANORAMICA SUL TI EXTENDED BASIC

Questo capitolo descrive brevemente i comandi del TI Extended BASIC, istruzioni e funzioni e suggerisce i modi con i quali questi possono essere usati.
Le prime otto sezioni riguardano Comandi, Assegnazioni ed Input, Output, Funzioni, Subroutines e Sottoprogrammi, Suono, Voce e Colori, Sprites, Debugging e trattamento degli errori. La sezione finale e' un esempio completo dell'immissione di un programma, che mostra i metodi di ingresso e l'uso di alcuni elementi del TI Extended BASIC.

**COMANDI**

I comandi dicono al computer di eseguire un compito non appena premuto **ENTER**, mentre le istruzioni vengono eseguite quando un programma e' in corso. In TI Extended BASIC molti comandi possono essere usati come istruzioni e la maggior parte delle istruzioni possono essere eseguite come comandi. Una lista di tutti i comandi, istruzioni e funzioni si trova nell'appendice B, ed indica i comandi che possono essere usati come istruzioni e le istruzioni che possono essere usate come comandi.

NEW
Per rimuovere un programma dal TI Extended BASIC al fine di preparare il computer ad accettare un nuovo programma, usare il comando NEW. I programmi sono rimossi dalla memoria anche con il comando OLD ed il comando RUN quando usato con un nome di file.

NUMBER e RESEQUENCE
Quando si sta inserendo un programma il computer assegna automaticamente il numero di linea se avete usato il comando NUMBER. Se desiderate rinumerare i numeri di linea di un programma dopo che questo e' stato scritto usare il comando RESEQUENCE. Formati abbreviati: NUM - RES.

LIST
Per rivedere il comando che e' stato immesso, usate il comando LIST. Il programma puo' essere listato sullo schermo o su una periferica collegata.

RUN
Il comando RUN ordina al computer di eseguire un programma. RUN puo' essere seguito da un numero di linea per fare in modo che l'esecuzione abbia inizio dal numero di linea specificato, o dal nome di un dispositivo di memoria di massa seguito dal nome file, per caricare e lanciare un programma da dischetto.

TRACE, UNTRACE, BREAK, UNBREAK e CONTINUE
Ciascuno di questi comandi riguarda la correzione di un programma, che consiste nel risolvere un problema che causa una condizione di errore o un risultato incorretto. Questi comandi sono spiegati ulteriormente nella sezione di questo capitolo riguardante la correzione ed il trattamento degli errori.

SAVE, OLD, MERGE e DELETE
Quando avete terminato di lavorare su un programma potreste aver bisogno di conservarlo su nastro o dischetto per un uso successivo. Il comando SAVE, seguito dal nome del dispositivo collegato e dal nome del programma, esegue questo compito. Percio' quando desiderate riutilizzare, listare, correggere o cambiare un programma, potete caricarlo in memoria con il comando OLD. Se un comando e' stato salvato usando l'opzione MERGE, potete fonderlo con un programma gia' presente in memoria utilizzando questo comando (MERGE). Quando non si ha piu' bisogno di un programma memorizzato su dischetto si puo' cancellarlo con il comando DELETE.

SIZE
Il comando SIZE vi permette di determinare quanto spazio libero e' rimasto in memoria, cosi' potete decidere se continuare ad aggiungere linee al programma o terminarlo ed avere un secondo programma che viene lanciato dal primo con l'uso di RUN come istruzione.

BYE
Quando avete terminato di usare il TI Extended BASIC, usate il comando BYE per tornare alla maschera principale.

Molti dei comandi (RUN, BREAK, UNBREAK, TRACE, UNTRACE e DELETE) possono essere usati nei programmi come istruzioni.

**ASSEGNAZIONI E INPUT**

Questo paragrafo tratta le istruzioni che nel TI Extended BASIC assegnano valori alle variabili e danno la possibilita' di immettere dati nei programmi.

LIST e READ
Se conoscete quali valori assegnare alle variabili usate LET e READ. Usate LET quando dovete assegnare una piccola quantita' di valori, o li state calcolando, e usate READ, in congiunzione con DATA e RESTORE, quando state assegnando una maggiore quantita' di valori.

INPUT e LINPUT
Quando desiderate che l'utilizzatore del programma immetta dei valori durante l'esecuzione, usando INPUT compare un prompt che chiede l'informazione necessaria. INPUT permette solo l'entrata dei valori all'estremita' inferiore dello schermo e non si puo' conoscere quale sia la natura del dato richiesto dal programma. L'ultima limitazione di INPUT e' che le virgole e gli apici intaccano cio' che viene immesso. Con LINPUT e' possibile inserire anche virgole ed apici all'interno delle stringhe da immettere. INPUT e LINPUT possono essere usati per leggere dati da files conservati su dischetti e cassette.

ACCEPT
ACCEPT permette l'inserimento dei dati da piu' parti dello schermo. Usando ACCEPT si elimina la necessita' di inserire i dati dall'estremita' inferiore dello schermo e lo "scrolling" che e' tipico dell'istruzione INPUT. Tuttavia ACCEPT non fa apparire alcun prompt. Quindi un'istruzione PRINT o DISPLAY deve essere inclusa nel programma per specificare il tipo di dati richiesto. ACCEPT puo' controllare che il dato inserito sia numerico, alfabetico, o ristretto a specifici caratteri. ACCEPT e' destinato solo all'uso di schermo e tastiera.

CALL KEY e CALL JOYST
Se la pressione di un singolo tasto e' tutto cio' che viene richiesto all'utilizzatore del programma, allora puo' essere usata l'istruzione CALL KEY. Per esempio, se il dato richiesto e' Y per "Yes" o N per "No" usate l'istruzione CALL KEY per accettare la loro immissione. Questa non fa apparire alcun carattere sullo schermo. Essa scandisce la tastiera o una parte di essa per verificare se e' stato premuto un tasto. La maggior limitazione della CALL KEY e' che accetta una singola pressione. Il dato introdotto non e' registrato come carattere, ma come il suo codice ASCII o altro codice (vedi l'appendice C e J per la lista dei codici usati). Se desiderate mostrare il carattere che e' stato premuto dovete usare DISPLAY, PRINT, CALL VCHAR o CALL HCHAR. L'inserimento da joystyck puo' essere effettuato con l'uso dell'istruzione CALL JOYST. Come con la CALL KEY il dato non e' visualizzato, e non c'e' alcuno "scrolling".

CALL CHARPAT, CALL COINC, CALL DISTANCE, CALL ERR, FOR-TO-STEP, CALL GCHAR, CALL POSITION, NEXT, CALL SPGET e CALL VERSION
Ciascuna di queste istruzioni assegna uno o piu' valori ad una variabile. CALL CHARPAT assegna un valore che specifica la forma (pattern) di un carattere. CALL COINC assegna un valore che verifica se gli sprites o uno sprite ed un punto sullo schermo sono vicini. CALL DISTANCE indica la distanza tra due sprites o tra uno sprite ed un punto sullo schermo. CALL ERR specifica l'errore che e' provocato e il punto ove si verifica. CALL GCHAR da' il carattere situato su un punto dello schermo. CALL POSITION da' la posizione di uno sprite sullo schermo. CALL SPGET assegna i codici dei valori di una frase ad una variabile per essere usata con CALL SAY. CALL VERSION indica la versione del BASIC usato.

FOR-TO-STEP e NEXT meritano un commento separato.
L'istruzione FOR-TO-STEP stabilisce il valore di una variabile in modo che essa possa controllare il numero di volte che un ciclo (loop) viene eseguito. Ogni volta che viene incontrato NEXT, il valore della variabile e' incrementato. Non appena il loop e' stato completato la variabile ha l'ultimo valore assunto dopo la fine del loop.

**OUTPUT**

Questa sezione tratta le istruzioni del TI Extended BASIC che sono usate per ottenere i risultati di un'elaborazione. Normalmente, l'output consiste nella visualizzazione di informazioni sullo schermo, stampa di dati, o salvataggio di dati su di un dispositivo esterno. Tuttavia l'output puo' anche essere usato per cambiare il colore dello schermo o dei caratteri, creare dei rumori o note, voce, o spedire dati alle periferiche.

PRINT, DISPLAY, PRINT...USING, DISPLAY...USING e IMAGE
Le istruzioni di output piu' frequentemente usate sono PRINT e DISPLAY. La punteggiatura (virgola, punto e virgola, due punti) e la funzione TAB sono usate per controllare la sistemazione dei dati in uscita. PRINT visualizza il dato all'estremita' inferiore dello schermo e ad ogni nuova visualizzazione scorre il vecchio dato di una riga verso l'alto. Con DISPLAY, si possono visualizzare dati in ogni parte dello schermo senza alcuno scorrimento. Con DISPLAY si puo' anche pulire lo schermo, cancellare i caratteri su una riga, ed emettere un beep.

PRINT...USING e DISPLAY...USING sono come PRINT e DISPLAY eccetto che il formato dei caratteri stampati o visualizzati e' determinato dalla clausola USING, meglio se unitamente ad un'istruzione IMAGE. La clausola USING permette l'esatto controllo del formato. PRINT e PRINT...USING, possibilmente in congiunzione con IMAGE, sono le sole istruzioni di output che possono essere usate per inviare dati ad una periferica (stampante o interfaccia seriale).

CALL HCHAR, CALL VCHAR e CALL SPRITE
CALL HCHAR e CALL VCHAR posizionano un carattere in una qualsiasi parte dello schermo e lo ripetono opzionalmente in orizzontale o in verticale. CALL SPRITE visualizza gli sprites sullo schermo. Gli sprites sono disegni che possono essere mossi in ogni direzione e cambiare forma, dimensioni e colori. CALL SPRITE e le altre istruzioni relative agli sprites sono discusse piu' avanti in questo capitolo.

CALL SCREEN e CALL COLOR
In aggiunta alla visualizzazione dei caratteri e dei dati sullo schermo e' possibile cambiare i loro colori. CALL SCREEN regola il colore dello schermo. CALL COLOR specifica il colore del carattere stesso (foreground) e del suo sfondo (background), dei caratteri e degli sprites.

CALL SOUND e CALL SAY
CALL SOUND fa emettere suoni. E' disponibile una vasta gamma di suoni. In piu', CALL SAY (possibimente usato con CALL SPGET) fa parlare il computer se ad esso e' collegato il sintetizzatore vocale.

**FUNZIONI, SUBROUTINES E SOTTOPROGRAMMI**

Il TI Extended BASIC fornisce funzioni estese e sottoprogrammi per il trattamento dei numeri e dei caratteri. In piu', potete costruire le vostre funzioni e scrivere i vostri sottoprogrammi e subroutines.
Le funzioni sono elementi del TI Extended BASIC che restituiscono un valore, normalmente basato su parametri assegnati alle funzioni. Molte funzioni sono di natura matematica; altre controllano o interessano il risultato o l'output prodotti dalle istruzioni nelle quali occorrono. Le funzioni del TI Extended BASIC sono: ABS, ASC, ATN, CHR$, COS, EOF, EXP, INT, LEN, LOG, MAX, MIN, PI, POS, REC, RND, RPT$, SEG$, SGN, SIN, SQR, STR$, TAB, TAN e VAL.

Potete anche definire le vostre funzioni usando DEF. Le funzioni sono usate all'interno delle istruzioni del TI Extended BASIC.

**Funzioni residenti**
Qui di seguito vengono spiegate brevemente queste funzioni.

| *Funzioni* | *Valori Restituiti e Commenti* |
|---|---|
| ABS | Valore assoluto di un'espressione numerica. |
| ASC | Il numero del codice ASCII del primo carattere di un'espressione di stringa. |
| ATN | Il valore dell'arcotangente di un'espressione numerica data in radianti. |
| CHR$ | Carattere corrispondente di un codice ASCII. |
| COS | Il valore del coseno di un'espressione numerica data in radianti. |
| EOF | End-Of-File. Condizione di fine del file. |
| EXP | Valore esponenziale di un'espressione numerica. |
| INT | Valore intero di un'espressione numerica. |
| LEN | Numero di caratteri contenuto in una stringa. |
| LOG | Logaritmo naturale di un'espressione numerica. |
| MAX | Il massimo valore tra due espressioni numeriche. |
| MIN | Il minimo valore tra due espressioni numeriche. |
| PI | P greco con un valore di 3.141592654 |
| POS | Posizione di un carattere incontrato per la prima volta all'interno di una stringa. |
| REC | Posizione di un record in un file. |
| RND | Numero casuale da 0 a 1. |
| RPT$ | Ripetizione di una stringa specifcata. |
| SEG$ | Segmento di una stringa che parte da un punto specificato e finisce dopo *n* caratteri. |
| SGN | Segno di un'espressione numerica. |
| SIN | Il valore del seno di un'espressione numerica data in radianti. |
| SQR | Radice quadrata di un'espressione numerica. |
| STR$ | Equivalente stringa di un'espressione numerica. |
| TAB | Posizione di stampa della successiva voce usata con PRINT, PRINT...USING, DISPLAY o DISPLAY...USING. |
| TAN | Il valore della tangente di un'espressione numerica data in radianti. |
| VAL | Equivalente numerico di una stringa contenente un numero. |

**Funzioni definite dall'utente**
DEF e' usato per definire le vostre funzioni. Le funzioni possono essere definite fino all'intera estensione di una linea fino ad un solo argomento. Le funzioni piu' lunghe possono essere costruite usando nuove funzioni che usano le funzioni precedentemente definite. Tuttavia, le funzioni piu' lunghe possono essere piu' efficacemente trattate con subroutines o sottoprogrammi.

**Subroutines**
GOSUB e ON...GOSUB sono usati per chiamare le subroutines. Una subroutine e' una serie di istruzioni destinate ad eseguire un compito piu' volte. Mediante l'uso di GOSUB o ON...GOUSUB non avete bisogno di digitare ogni volta che occorrono le stesse linee. La subroutine puo' usare i valori di ciascuna variabile del programma e cambiarne il valore ogni volta che viene eseguita.

**Sottoprogrammi residenti**
I sottoprogrammi residenti sono elementi del TI Extended BASIC che eseguono particolari funzioni. Ad essi vi si accede sempre con l'istruzione CALL, seguita dal nome del sottoprogramma. Essi sono: CHAR, CHARPAT, CHARSET, CLEAR, COINC, COLOR, DELSPRITE. I sottoprogrammi residenti sono elementi del TI Extended BASIC che eseguono particolari funzioni. Ad essi vi si accede sempre con l'istruzione CALL, seguita dal nome del sottoprogramma. Essi sono: CHAR, CHARPAT, CHARSET, CLEAR, COINC, COLOR, DELSPRITE, DISTANCE, ERR, GCHAR, HCHAR, INIT, JOYST, KEY, LINK, LOAD, LOCATE, MAGNIFY, MOTION, PATTERN, PEEK, POSITION, SAY, SCREEN, SOUND, SPGET, SPRITE, VCHAR e VERSION.

I sottoprogrammi predefiniti eseguono svariati compiti. Alcuni dei sottoprogrammi riguardano la visualizzazione e stabiliscono se qualche tasto e' stato premuto sulla tastiera.

| *Sottoprogrammi Predefiniti* | *Azione e Commenti* |
|---|---|
| CLEAR | Pulisce lo schermo. |
| COLOR | Determina i colori dei caratteri del set-caratteri o il colore degli sprites. |
| GCHAR | Restituisce il codice ASCII del carattere situato in una determinata posizione dello schermo. |
| HCHAR | Visualizza un carattere, e opzionalmente lo ripete in orizzontale. |
| JOYST | Restituisce i valori che indicano la posizione dei joysticks. |
| KEY | Restituisce il codice del tasto premuto. |
| SCREEN | Specifica il colore dello schermo. |
| VCHAR | Visualizza un carattere sullo schermo e opzionalmente lo ripete in verticale. |

I sottoprogrammi predefiniti possono definire e controllare anche gli sprites.

| *Sottoprogrammi Predefiniti* | *Azione e Commenti* |
|---|---|
| CHAR | Specifica la forma di un carattere usato come sprite o come disegno. |
| COINC | Determina se due sprites o uno sprite ed un punto sullo schermo sono nella stessa locazione dello schermo o in quella vicina. |
| COLOR | Specifica il colore di uno sprite o di un set di caratteri. |
| DELSPRITE | Cancella gli sprites. |
| DISTANCE | Determina la distanza tra due sprites o tra uno sprite e un punto sullo schermo. |
| LOCATE | Specifica la posizione di uno sprite. |
| MAGNIFY | Cambia la dimensione degli sprites. |
| MOTION | Specifica direzione e velocita' di uno sprite sullo schermo. |
| PATTERN | Specifica il carattere che definisce uno sprite. |
| POSITION | Determina la posizione di uno sprite. |
| SPRITE | Definisce gli sprites, specificando il carattere che li definisce, il loro colore, posizione e movimento. |

Una terza categoria di sottoprogrammi predefiniti in TI Extended BASIC comprende il suono e la voce.

| *Sottoprogrammi Predefiniti* | *Azione e Commenti* |
|---|---|
| SAY | Fa in modo che il computer parli se ad esso e' collegato il sintetizzatore vocale. |
| SOUND | Genera i suoni. |
| SPGET | Recupera i codici che generano la voce. |

Altri quattro sottoprogrammi predefiniti servono soltanto per l'uso in linguaggio macchina del microprocessore TMS9900. Istruzioni dettagliate sull'uso di questi sottoprogrammi: INIT, LINK, LOAD, e PEEK vengono fornite con i sottoprogrammi del linguaggio macchina.

Infine ci sono alcuni sottoprogrammi predefiniti di tipo misto.

| *Sottoprogrammi Predefiniti* | *Azione e Commenti* |
|---|---|
| CHARPAT | Restituisce un valore che identifica la sagoma di un carattere. |
| CHARSET | Riporta i caratteri da 32 a 95 alle loro sagome e colori predefiniti. |
| ERR | Restituisce dei valori che danno informazioni sul tipo di errore eventualmente occorso. |
| VERSION | Specifica la versione del BASIC usato. |

**Sottoprogrammi autocostruiti**

Potete scrivere i vostri sottoprogrammi. Essi sono una serie di istruzioni designate per eseguire un compito. Possono essere usati in un programma quando e' richiesta piu' volte l'esecuzione dello stesso compito in vari programmi diversi. Se utilizzate l'opzione MERGE quando salvate un sottoprogramma, avrete la possibilita' di includerlo in altri programmi.

Quando un sottoprogramma si trova in un programma, esso deve seguire il programa principale. La struttura di un programma deve essere come segue:

| | |
|---|---|
| Inizio programma principale | |
| - | |
| - | |
| - | |
| Chiamate sottoprogramma | |
| - | |
| - | |
| - | |
| Fine del programma principale | Il programma si fermera' qui senza un'istruzione STOP o END. |
| Inizio primo sottoprogramma | I sottoprogrammi sono opzionali. |
| - | |
| - | |
| - | |
| Fine del primo sottoprogramma | Non deve apparire niente tra i sottoprogrammi ec_ cetto istruzioni REM e END. |
| Inizio secondo sottoprogramma | |
| - | |
| - | |
| - | |
| Fine secondo sottoprogramma | Soltanto commenti ed END possono apparire dopo i sottoprogrammi. |
| Fine del programma | |

I sottoprogrammi sono chiamati mediante l'uso di CALL seguito dal nome del sottoprogramma e da una lista facoltativa di parametri e valori. La prima linea di un sottoprogramma e' SUB, seguita dal nome del sottoprogramma ed opzionalmente da una lista di parametri.

I sottoprogrammi che scrivete non fanno parte del programma principale. Essi non usano i valori delle variabili del programma prinicipale, cosicche' ogni valore di cui si ha bisogno deve essere dotato di una lista di parametri specificata nell'istruzione CALL. I nomi di variabile possono anche essere uguali a quelli delle variabili del programma principale o di altri sottoprogrammi, senza per questo influenzarne i valori. I sottoprogrammi possono chiamare altri sottoprogrammi, ma non devono chiamare se stessi, ne' direttamente ne' indirettamente.

SUBEND deve essere l'ultima istruzione in un sottoprogramma. Quando quest'istruzione e' eseguita, il controllo ritorna all'istruzione successiva a quella che ha chiamato il sottoprogramma. Si puo' uscire da un sottoprogramma anche con l'istruzione EXIT.

## SUONO VOCE e COLORE

Potete mettere in risalto particolari sezioni dell'uscita del vostro programma mediante l'uso di suoni, voce e colori. Questa "umanizzazione" del computer rende il programma piu' facile ed interessante da usare.

CALL SOUND
SOUND da' l'uscita di suoni. I toni possono essere prodotti con durata variante da .001 a 4.25 secondi e volume da 0 (il piu' forte) a 30 (il piu' debole). La gamma di frequenza varia da 110 (livello al di sotto del LA) a 44733, oltre la portata dell'udito umano. Inoltre sono disponibili 8 diversi rumori. Possono essere prodotti fino a 3 toni ed un rumore contemporaneamente. L'Appendice D elenca le frequenze usate per la produzione di toni musicali.

CALL SAY e CALL SPGET
SAY fa emettere parole quando il sintetizzatore vocale e' collegato alla consolle. Potete scegliere tra 373 lettere, numeri, parole e frasi (elencati nell'Appendice L). Tra l'altro potete costruire nuove parole utilizzando la combinazione di quelle residenti. Per esempio, SOME+THING produce "SOMETHING" e THERE+FOUR produce "THEREFORE".

SPGET e' usato per restituire i codici vocali che producono la voce. Questa configurazione puo' quindi essere usata per produrre linguaggi piu' naturali e per cambiare parole. Poiche' la creazione di nuove parole e' un processo complicato, non e' discussa in questo manuale. Comunque, i suffissi possono essere aggiunti abbastanza semplicemente.
L'Appendice M spiega come aggiungere i suffissi ING, S, e ED ad ogni parola cosicche' parole come ANSWERING, ANSWERS, ANSWERED, INSTRUCTING, e INSTRUCTED vengano incluse nel vocabolario del computer.

CALL COLOR e CALL SCREEN
COLOR cambia i colori del set dei caratteri e determina i colori degli sprites. SCREEN specifica il colore dello schermo tra uno dei 16 colori disponibili sul TI-99/4A.

## SPRITES

Gli sprites sono disegni che possono essere visualizzati e mossi sullo schermo. Un vantaggio che gli sprites hanno rispetto agli altri caratteri e' che essi possono essere posizionati in ciascuna delle 49152 posizioni (192 righe X 256 colonne) invece che nelle 768 di 24 righe e 32 colonne usate dalle istruzioni CALL VCHAR e CALL HCHAR. Il motivo di questa alta risoluzione, e' che gli sprites si possono muovere in modo piu' agevole dei caratteri. Inoltre, una volta iniziato il loro movimento, gli sprites possono continuare a muoversi senza ulteriore controllo del programma.

CALL SPRITE
CALL SPRITE definisce gli sprites. Questo sottoprogramma specifica la forma usata dagli sprites, il loro colore, la loro posizione, e, opzionalmente, il loro movimento.

CALL CHAR e CALL MAGNIFY
Sebbene possiate utilizzare ciascuno dei caratteri predefiniti, quelli disponibili tra 32 e 95, come uno sprite, CALL CHAR e' generalmente usato per definire la nuova forma di uno sprite. Per formare uno sprite puo' essere usata una matrice di 8x8 punti (pixels) che puo' essere ingrandita fino a 4 volte. Il sottoprogramma MAGNIFY controlla la risoluzione e la dimensione degli sprites.

CALL COLOR, CALL LOCATE, CALL PATTERN, e CALL MOTION
Non appena e' stato creato uno sprite, esso puo' essere modificato da altri sottoprogrammi. COLOR ne cambia il colore. LOCATE lo sposta in un'altra posizione. PATTERN cambia il carattere che lo definisce. MOTION modifica il suo movimento.

CALL COINC, CALL DISTANCE e CALL POSITION
Tre sottoprogrammi forniscono informazioni sugli sprites mentre un programma e' in esecuzione. COINC restituisce un valore che indica se gli sprites o uno sprite ed un punto sullo schermo sono vicini o nello stesso punto dello schermo. DISTANCE restituisce un valore che specifica la distanza tra due sprites o uno sprite ed un punto sullo schermo. POSITION riporta dei valori che indicano la posizione di uno sprite.

CALL DELSPRITE
CALL DELSPRITE permette di cancellare gli sprites. Se preferite, potete nascondere gli sprites posizionandoli fuori dalla portata dello schermo.

**DEBUGGING**

Fare il debug di un programma significa trovare errori logici o di sintassi all'interno di esso.
BREAK, CONTINUE, TRACE, ON BREAK, UNBREAK, UNTRACE e **FCTN 4** (CLEAR) sono spesso usati nella ricerca degli errori.

BREAK, ON BREAK, CONTINUE e UNBREAK
BREAK causa l'interruzione di un programma, in modo che possiate farvi stampare i valori delle variabili ed eventualmente modificarli. BREAK riporta anche ai loro valori standard i colori dei caratteri (neri su trasparente) ed il colore standard dello schermo, restituisce ai caratteri da 32 a 95 la loro rappresentazione di default, ed elimina gli sprites.

ON BREAK dice al computer cosa fare in caso di interruzione del programma. Potete usare questa istruzione per dire al computer di ignorare i "breakpoints" che avete immesso nel programma. CONTINUE (CON) ordina al computer di continuare l'esecuzione del programma interrotto. UNBREAK annulla tutti i "breakpoints" inseriti con BREAK. *Nota:* se avete inserito ON BREAK NEXT, il computer non si arrestera' alla pressione dei tasti **FCTN 4** (CLEAR).

TRACE e UNTRACE
TRACE fa in modo che il computer visualizzi ogni numero di linea durante l'esecuzione del programma. Usando questa istruzione avrete la possibilita' di seguire la sequenza delle operazioni di un programma. UNTRACE annulla l'effetto di TRACE.

**TRATTAMENTO DEGLI ERRORI**

Potete includere delle istruzioni in un programma che indicano quale tipo di errore e' occorso durante la sua esecuzione.

CALL ERR, ON ERROR, ON WARNING e RETURN
CALL ERR vi da' informazioni che indicano dove e' avvenuto un errore e che tipo di errore e'. L'Appendice N elenca i codici dei vari tipi di errore. ON ERROR specifica cosa fa il computer se avviene un errore. ON WARNING specifica cosa fa il computer se si presenta la condizione che normalmente provocherebbe la visualizzazione di un messagio di errore. RETURN e' usato con ON ERROR oltre che con GOSUB. Esso ripete l'esecuzione dell'istruzione che ha causato l'errore, ritorna all'istruzione seguente a quella che ha causato l'errore, o trasferisce il controllo ad un'altra parte del programma che evita il ripetersi dell'errore.

**ESEMPIO D'INTRODUZIONE DI UN PROGRAMMA**

Adesso che abbiamo esaminato brevemente le caratteristiche del TI Extended BASIC, possiamo divertirci a correggere o anche a cominciare a fare esperimenti su un programma dimostrativo. Questo paragrafo dimostra alcune delle caratteristiche piu' utili del TI Extended BASIC. Seguendo i consigli di questo paragrafo potrete imparare ad apprendere qualche utile scorciatoia nella programmazione.

Questo programma vi permette di fare un gioco chiamato Codebreaker. Giocandoci potete determinare la lunghezza di un codice da 1 a 8 cifre. Quindi decidete la gamma delle cifre che possono essere incluse nel codice (fino a 10). Il computer seleziona le cifre del codice senza ripeterle. Voi dovete cercare di indovinare quali sono le cifre e la loro sequenza. Dopo ogni ipotesi il computer vi dice quante cifre avete indovinato e quante sono nel posto giusto. Se ripetete una cifra nella vostra risposta, essa e' contata come giusta ogni volta che compare. Approfittando di questa informazione, provate di nuovo. Si vince quando indovinate correttamente la sequenza.

Per esempio, supponete di aver scelto di giocare usando 4 cifre ognuna delle quali sia di 9 numeri (0,1,2,3,4,5,6,7 o 8). Il codice che il computer sceglie puo' essere 0743, che state provando a decifrare. In questo caso esiste una possibile sequenza di ipotesi.

| *IPOTESI* | *GIUSTA* | *POSIZ.* | *SPIEGAZIONE DELLA RISPOSTA DEL COMPUTER* |
|---|---|---|---|
| 0000 | 4 | 1 | 0 e' giusto 4 volte, e 1 anche come posto |
| 1234 | 2 | 0 | 3 e 4 giusti ma non nel posto giusto |
| 5678 | 1 | 0 | 7 e' giusto ma non nel posto giusto |
| 2348 | 2 | 1 | 3 e 4 sono giusti e 4 e' nel posto giusto |
| 0347 | 4 | 2 | tutto giusto, 0 e 4 nel posto giusto |
| 3047 | 4 | 1 | tutto giusto, il 4 nel posto giusto |
| 0734 | 4 | 2 | tutto giusto, 0 e 7 nel posto giusto |
| 0743 | 4 | 4 | tutto giusto e nella giusta sequenza Avete vinto. |

Prima di cominciare ad inserire il programma, accendere tutte le periferiche che avete collegate al computer. Inserire il modulo TI Extended BASIC ed accendere anche il computer. Premere un tasto qualsiasi per andare alla lista di selezione principale. Premere 2 per selezionare il TI Extended BASIC.

Nelle pagine che seguono, i caratteri che digitate ed i tasti che premete sono SOTTOLINEATI.

## CODEBREAKER

| Commenti | Schermo | |
|---|---|---|
| | * READY * | |
| Numera automaticamente le linee del programma. | >NUM | ENTER |
| Titolo e linguaggio. | >100 REM CODEBREAKER XBASIC | ENTER |
| Riserva spazio per i codici e le risposte. | >110 DIM CODE$(8),GUESS$(8) | ENTER |
| Crea codici a caso. | >120 RANDOMIZE | ENTER |
| Pulisce lo schermo, suona,e mette il titolo CODEBREAKER a partire dalla 9.a colonna sulla 11.a riga. | >130 DISPLAY AT(11,9)BEEP ERA SE ALL:"CODEBREAKER" | ENTER |
| REDO ripete tutto cio' che e' stato digitato prima di premere **ENTER**. Usando i tasti di edit (**FCTN 2** (INSERT), **FCTN 1** (DELETE) e le frecce), cambia la linea 130 in: 140 DISPLAY AT(19,1) BEEP: "NUMBER OF CODES? (1-8)". | >140 | FCTN 8 |
| Suona e visualizza NUMBER OF CODES? (1-8) a partire dalla 1.a colonna sulla 19.a riga. | 140 DISPLAY AT(19,1 )BEEP: "NUMBER OF CODES? (1-8)" | ENTER |
| Premere di nuovo **FCTN 8**. Ora cambia la riga 140 in: 150 DISPLAY AT(21,6)BEEP: "DIGITS EACH CODE?". | > | FCTN 8 |
| Suona, e visualizza DIGITS EACH CODE? a partire dalla 6.a colonna sulla 21.a riga. | 150 DISPLAY AT(21,6 )BEEP: "DIGITS EACH CODE?" | ENTER |
| Accetta a riga 19 e colonna 24 l'immisssione dei codici, ristrettamente alle cifre ammesse in CODES. | >160 ACCEPT AT(19,24)VALIDATE (DIGIT):CODES | ENTER |
| Cambia la linea 160 in: 170 ACCEPT AT(21,24)VALIDATE (DIGIT):DIGITS. | > | FCTN 8 |
| Accetta a riga 21 e colonna 24 l'immissione dei codici ristrettamente alle cifre ammesse in DIGITS. | 170 ACCEPT AT(21, 24)VALIDATE (DIGIT):DIGITS | ENTER |

Visualizza il programma come e' stato inserito.

```
>LIST
 100 REM CODEBREAKER XBASIC
 110 DIM CODE$(8),GUESS$(8)
 120 RANDOMIZE
 130 DISPLAY AT(11,9)BEEP ERASE ALL:
 "CODEBREAKER"
 140 DISPLAY AT(19,1)BEEP:"NUMBER OF
 CODES? (1-8)"
 150 DISPLAY AT(21,6)BEEP:"DIGITS
 EACH CODE?"
 160 ACCEPT AT(19,24)VALIDATE(DIGIT)
 :CODES
 170 ACCEPT AT(21,24)VALIDATE(DIGIT)
 :DIGITS
```

Lancia il programma. Lo schermo si pulisce e appare questo:

```
>RUN

            CODEBREAKER


   NUMBER OF CODES? (1-8) ■

            DIGITS EACH CODE?
```

Introducete qualsiasi carattere che non sia un numero, il computer emettera' un suono e non lo accettera'. Introducete 4. Il cursore si spostera' sotto al precedente. Introducete 10, il programma termina e potete continuare l'immissione di altre linee.

```
* READY *
>NUM 180                                        ENTER
```

Numera le linee a partire da 180.

```
>180 IF CODES>DIGITS THEN DIS
 PLAY AT(24,2)BEEP:"NO MORE C
 ODES THAN DIGITS"::GOTO 160                    ENTER
```

Controllo sulla quantita' di digits inseriti. Se CODES e' minore o uguale a DIGITS, il controllo passa alla linea successiva. Se CODES e' maggiore di DIGITS, sull'ultima linea dello schermo appare il messaggio: NO MORE CODES THAN DIGITS, il controllo viene trasferito alla linea 160.

```
>190 FOR A=1 TO CODES !CHOOSE
 CODES                                          ENTER
```

Inizio del ciclo per la scelta dei codici. Le parole dopo ! sono commenti.

```
>200 CODE$(A)=STR$(INT(RND*DI
 GITS))                                         ENTER
```

Sceglie i codici a caso.

| | | |
|---|---|---|
| Inizio del ciclo che evita la duplicazione dei codici. | >210 FOR_B=0_TO_A-1_!CHECK_FOR_DUPLICATES | ENTER |
| Controlla i duplicati. Sceglie un nuovo codice, se ce n'e' uno duplicato. | >220 IF_CODE$(A)=CODE$(B)_THEN_200 | ENTER |
| Termine del ciclo del controllo. | >230 NEXT_B | ENTER |
| Fine del ciclo per la scelta del codice. | >240 NEXT_A | ENTER |
| Assegna una variabile per la posizione sullo schermo del messaggio. | >250 ROW=2 | ENTER |
| Pulisce lo schermo e fa ap_parire l'intestazione. | >260 DISPLAY_AT(1,1)ERASE_ALL:"GUESS_____RIGHT___PLACE" | ENTER |
| Con il REDO della linea 260 avremo: 270 DISPLAY AT(24,3):"ENTER 'X' FOR SOLUTION". | >270 | FCTN 8 |
| Visualizza un'istruzione al margine inferiore dello schermo. | 270 DISPLAY AT(24,3):"ENTER 'X'_FOR_SOLUTION" | ENTER |
| Numera le righe a partire da 280. | >NUM_280 | ENTER |
| Accetta la risposta alla riga appropriata. | >280 ACCEPT_AT(ROW,1):C$ | ENTER |
| Controllo per decidere se continuare o ricominciare. | >290 IF_C$="X"_THEN_470_!GIVE UP_OR_RESET | ENTER |
| Inizia il ciclo che pone fine alla risposta per controllarla con precisione. | >300 FOR_D=1_TO_CODES_!BREAK UP_GUESS | ENTER |
| Separa la risposta data in singole cifre. | >310 GUESS$(D)=SEG$(C$,D,1) | ENTER |
| Termina il ciclo per separare la risposta. | >320 NEXT_D | ENTER |
| Pone a zero le variabili RIGHT e PLACE. | >330 RIGHT,PLACE=0 | ENTER |
| Inizia un ciclo esterno per confrontare la risposta con il codice. | >340 FOR_E=1_TO_CODES_!CHECK GUESS_FOR_CORRECTNESS | ENTER |
| Inizia il ciclo interno per controllare la risposta. | >350 FOR_F=1_TO_CODES | ENTER |
| Se una risposta non corrisponde al codice va alla linea successiva, altrimenti aggiunge 1 alla variabile contenente la risposta esatta (RIGHT). Poi se la risposta e' al posto giusto aggiunge 1 alla variabile che contiene la posizione giusta (PLACE). | >360 IF_CODE$(E)=GUESS$(F)_THEN_RIGHT=RIGHT+1::IF_E=F_THEN_PLACE=PLACE+1 | ENTER |

| | | |
|---|---|---|
| Completa il ciclo interno. | >370 NEXT_E | **ENTER** |
| Completa il ciclo esterno. | >380 NEXT_E | **ENTER** |
| Visualizza il numero di cifre che sono corrette. | >390 DISPLAY_AT(ROW,14):RIGHT | **ENTER** |
| REDO alla linea 390: 400 DISPLAY AT(ROW,22):PLACE. | >400 | **FCTN 8** |
| Visualizza il numero di cifre che sono al posto giusto. | 400 DISPLAY AT(ROW,22):PLACE | **ENTER** |
| Numera le linee a partire da 410. | >NUM_410 | **ENTER** |
| Controlla se il codice e' stato risolto, se lo e' stato va alla linea successiva. Altrimenti aggiunge 1 alla riga (ROW). Poi se la riga e' maggiore di 22, va alla linea 470 e visualizza la soluzione. Altrimenti ritorna alla linea 280 per accettare una nuova risposta. | >410 IF_PLACE<>CODES_THEN_ROW =ROW+1::IF_ROW>22_THEN_470_E LSE_280 | **ENTER** |
| Visualizza il messaggio di vittoria con il numero di risposte a riga 23 e colonna 1. | >420 DISPLAY_AT(23,1)BEEP:"YO U_WIN_WITH";ROW-1;"GUESSES." | **ENTER** |
| REDO alla linea 420: 430 DISPLAY AT(24,1)BEEP:"PLAY AGAIN? (Y/N) Y". | >430 | **FCTN 8** |
| Visualizza il prompt PLAY AGAIN? (Y/N) Y a riga 24 colonna 1. | 430 DISPLAY AT(24 ,1)BEEP:"PL AY_AGAIN?_(Y/N)_Y" | **ENTER** |
| Numera le linee a partire da 440. | >NUM_440 | **ENTER** |
| Accetta un ingresso in X$ a riga 24 e colonna 19. Non sposta nessun carattere che si trova gia' in quel punto (in questo caso, la Y della istruzione DISPLAY alla linea 430), accetta solo un carattere limitato alla Y o alla N ed emette un BEEP. Premendo **ENTER** a questo punto si conferma la Y che e' gia' presente dalla linea 430. | >440 ACCEPT_AT(24,19)SIZE(-1) BEEP_VALIDATE("YN"):X$ | **ENTER** |
| Se inserita la Y, si ritorna alla linea 190 e il gioco ricomincia. | >450 IF_X$="Y"_THEN_190 | **ENTER** |
| Fine del programma. | >460 STOP | **ENTER** |

Prima di eseguire un programma e' opportuno correggerlo. Qui di seguito e' riportato l'intero programma digitato fin'ora per permettervi di controllare ancora il listato del programma.

```
100 REM CODEBREAKER XBASIC
110 DIM CODE$(8),GUESS$(8)
120 RANDOMIZE
130 DISPLAY AT(11,9)BEEP ERA
SE ALL:"CODEBREAKER"
140 DISPLAY AT(19,1)BEEP:"NU
MBER OF CODES? (1-8)"
150 DISPLAY AT(21,6)BEEP:"DI
GITS EACH CODE?"
160 ACCEPT AT(19,24)VALIDATE
(DIGIT):CODES
170 ACCEPT AT(21,24)VALIDATE
(DIGIT):DIGITS
180 IF CODES>DIGITS THEN DIS
PLAY AT(24,2)BEEP:"NO MORE C
ODES THAN DIGITS" :: GOT
O 160
190 FOR A=1 TO CODES !CHOOSE
 CODES
200 CODE$(A)=STR$(INT(RND*DI
GITS))
210 FOR B=0 TO A-1 !NO DUPLI
CATES
220 IF CODE$(A)=CODE$(B)THEN
 200
230 NEXT B
240 NEXT A
250 ROW=2
260 DISPLAY AT(1,1)ERASE ALL
:"GUESS      RIGHT    PLACE"
270 DISPLAY AT(24,3):"ENTER
'X' FOR SOLUTION"
280 ACCEPT AT(ROW,1):C$
290 IF C$="X" THEN 470 !GIVE
 UP OR RESET
300 FOR D=1 TO CODES !BREAK
```

```
UP GUESS

310 GUESS$(D)=SEG$(C$,D,1)
320 NEXT D
330 RIGHT,PLACE=0
340 FOR E=1 TO CODES !CHECK
GUESS
350 FOR F=1 TO CODES
360 IF CODE$(E)=GUESS$(F)THE
N RIGHT=RIGHT+1 :: IF E=F TH
EN PLACE=PLACE+1
370 NEXT F
380 NEXT E
390 DISPLAY AT(ROW,14):RIGHT
400 DISPLAY AT(ROW,22):PLACE
410 IF PLACE<>CODES THEN ROW
=ROW+1 :: IF ROW>22 THEN 470
 ELSE 280
420 DISPLAY AT(23,1)BEEP:"YO
U WIN WITH";ROW-1;"GUESSES."
430 DISPLAY AT(24,1)BEEP:"PL
AY AGAIN? (Y/N) Y"
440 ACCEPT AT(24,19)SIZE(-1)
BEEP VALIDATE("YN"):X$
450 IF X$="Y" THEN 190
460 STOP
470 DISPLAY AT(23,1)BEEP:"TH
E CODE IS" !LOSE, GIVE UP OR
 RESET
480 FOR G=1 TO CODES
490 DISPLAY AT(23,12+G):CODE
$(G)
500 NEXT G
510 DISPLAY AT(24,1)BEEP:"PL
AY AGAIN? (Y/N) Y"
520 ACCEPT AT(24,19)SIZE(-1)
BEEP VALIDATE("YN"):X$
530 IF X$="Y" THEN 130
```

Ora lanciate il programma digitando RUN seguito da **ENTER.** Scegliere 4 numeri di codice con 10 cifre (0,1,2,3,4,5,6,7,8 e 9) possibili per ciascun codice. Indovinare in 6 risposte e' eccellente. Indovinare il codice in 8 risposte e' molto buono.

Se desiderate usare di nuovo il programma salvatelo su dischetto o cassetta. Per registrarlo su cassetta assicurarsi che il registratore sia collegato. Poi digitare SAVE CS1 e seguire le istruzioni che compaiono sullo schermo.

Per registrare il programma su disco digitare SAVE DSK1.*Nome-file* con qualsiasi *Nome-file* da voi desiderato, come ad esempio CODEBREAK.

Dopo aver salvato il programma, o se non desiderate salvarlo, digitare NEW. Il programma e' cancellato dalla memoria del computer e ne potete digitare un altro.

Se avete salvato il programma, potete facilmente caricarlo nella memoria del computer per un suo riutilizzo o per revisioni, correzioni o aggiunte. Ricaricare il programma da cassetta con il comando OLD CS1 e seguire le istruzioni successive che compaiono sullo schermo.
Ricaricare il programma da dischetto con il comando OLD DSK1.*Nome-file* usando il nome che gli avevate assegnato.

Quando avete terminato di usare il TI Extended BASIC digitare BYE per tornare alla maschera di accensione.

# CAPITOLO 3

## REGOLE DEL TI EXTENDED BASIC

================================================================
================================================================
================================================================

Questo capitolo tratta il formato che devono avere i programmi in TI Extended BASIC ed il modo nel quale esso funziona.

## PARTENZA AUTOMATICA DI UN PROGRAMMA

Quando viene selezionato il TI Extended BASIC, se nel dischetto situato nel Disk Drive 1 si trova un programma chiamato LOAD, quel programma sara' automaticamente caricato ed eseguito. L'effetto e' lo stesso che si ha con RUN "DSK1.LOAD". Se il programma non esiste, ci sara' una pausa momentanea mentre il TI Extended BASIC lo cerca.

## FILES

I files sono gruppi di dati conservati su periferiche esterne, chiamate "memorie di massa". I files piu' comuni sono conservati su cassette o dischetti, ma per il TI Extended BASIC sono considerati files anche i dati inviati attraverso altre periferiche esterne come l'interfaccia RS232 o la stampante.

## NUMERI DI LINEA

Il TI Extended BASIC richiede numeri di linea per i suoi programmi. I numeri di linea specificano l'ordine nel quale vengono eseguite le linee e vengono usati per identificare quali sono le righe da seguire quando si usano: IF-THEN-ELSE, GOTO, GOSUB, ON ERROR, ON...GOTO e ON...GOSUB. I numeri di linea possono anche essere usati con BREAK, LIST, NUM, RESTORE, RETURN e RUN. I numeri di linea devono essere numeri interi compresi tra 1 e 32767.

Il computer genera automaticamente i numeri di linea se usate il comando NUM. Quando questo non e' seguito da un numero di linea il computer incrementera' i numeri di linea di 10 in 10 a partire da 100. E' possibile rinumerare le linee con il comando RES.

## LINEE

Le linee possono essere lunghe fino a 140 caratteri, incluso il numero di linea e gli spazi. Se siete arrivati alla fine di una linea, gli ultimi caratteri che provate ad aggiungere vengono riposizionati sul carattere 140. E' possibile creare una linea piu' lunga di 140 caratteri in Modo Edit usando **FCTN 2** (INSERT).

## SIMBOLI SPECIALI

Simboli speciali separano le istruzioni dai commenti su una stessa linea. Una linea del TI Extended BASIC e' composta da un numero di linea, una o piu' istruzioni ed un commento facoltativo. Per esempio:

```
100 FOR A=1 TO 100 :: PRINT A;SQR(A) :: NEXT A !STAMPA LE RADICI
QUADRATE
```

Il simbolo separatore di istruzione, due volte due punti (::), e' usato per separare le istruzioni su una stessa linea. Il punto esclamativo (!) e' usato per separare una nota esplicativa dal resto di una linea. Il computer non tiene alcun conto dei commenti durante l'esecuzione del programma.

**SPAZI**

Gli spazi sono richiesti dal TI Extended BASIC tra gli elementi che compongono le istruzioni, per permettere al computer di distinguere i nomi delle variabili dagli elementi del TI Extended BASIC. Tuttavia gli spazi non sono richiesti ne' prima ne' dopo i simboli relazionali, dei punti esclamativi o dei simboli separatori. Potete inserire spazi aggiuntivi durante l'inserimento di comandi e di istruzioni, ma essi sono eliminati dal TI Extended BASIC. Quando i programmi vengono listati, il TI Extended BASIC puo' aggiungere spazi prima e dopo il punto esclamativo ed i simboli separatori.

**COSTANTI NUMERICHE**

Le costanti numeriche possono essere inserite con qualsiasi numero di cifre. Tuttavia esse sono arrotondate a 13 o 14 cifre dal computer per permettergli di immagazzinarle nel suo formato interno, e sono generalmente visualizzate fino ad un massimo di 10 cifre. Per numeri estremamente grandi o piccoli, e' in genere piu' conveniente usare la notazione scientifica. In genere il computer usa la notazione scientifica quando stampa numeri molto grandi o molto piccoli.

Nella notazione scientifica, un numero e' dato come una mantissa (un numero significativo a sinistra del punto decimale) moltiplicato 10 elevato a potenza di un numero intero. 15 e' espresso in notazione scientifica come $1.5 \times 10^{1}$. 150 e' espresso come $1.5 \times 10^{2}$; -1500 e' espresso come $-1.5 \times 10^{3}$; 156.789.000.000.000 e' espresso come $1.56789 \times 10^{14}$; e 0.156789 e' espresso come $1.56789 \times 10^{-1}$. In TI Extended BASIC il "x 10" e' rappresentato da "E". Cosi' $1.5 \times 10^{3}$ diventa 1.5E3.

Le costanti numeriche sono definite nella gamma da -9.9999999999999E127 a -1E-128, 0, e da 1E-128 a 9.9999999999999E127. Se l'esponente di un numero calcolato e' maggiore o minore di 99, "**" e' generalmente stampato o visualizzato come potenza. L'esponente intero e' conservato internamente e puo' essere visualizzato con la clausola USING in un'istruzione PRINT o DISPLAY.

**STRINGHE**

Le stringhe nel TI Extended BASIC possono arrivare alla lunghezza di una linea di schermo (28 caratteri). Se nella stringa e' compresa un'altra coppia di apici essa verra' considerata come una doppia stringa.

**VARIABILI**

I nomi delle variabili possono essere composti da 1 a 15 caratteri. Il primo carattere di una variabile deve essere una lettera dell'alfabeto, il simbolo della chiocciola, o il sottolineato (_). I caratteri successivi possono essere quei simboli in aggiunta ad ognuna delle cifre. L'ultimo carattere di una variabile stringa deve essere sempre il segno del dollaro ($). Le variabili possono essere sia semplici che matrici fino a sette dimensioni.

Certe parole sono riservate all'uso del TI Extended BASIC. Esse sono comandi, istruzioni, funzioni ed operatori che fanno parte del linguaggio. Queste parole non possono essere usate come nomi di variabili, ma possono farne parte. La seguente e' una lista completa delle parole riservate al TI Extended BASIC.

| | | | |
|---|---|---|---|
| ABS | EOF | NUMBER | SEQUENTIAL |
| ACCEPT | ERASE | NUMERIC | SGN |
| ALL | ERROR | OLD | SIN |
| AND | EXP | ON | SIZE |
| APPEND | FIXED | OPEN | SQR |
| ASC | FOR | OPTION | STEP |
| AT | GO | OR | STOP |
| ATN | GOSUB | OUTPUT | STR$ |
| BASE | GOTO | PERMANENT | SUB |
| BEEP | IF | PI | SUBEND |
| BREAK | IMAGE | POS | SUBEXIT |
| BYE | INPUT | PRINT | TAB |
| CALL | INT | RANDOMIZE | TAN |
| CHR$ | INTERNAL | READ | THEN |
| CLOSE | LEN | REC | TO |
| CON | LET | RELATIVE | TRACE |
| CONTINUE | LINPUT | REM | UALPHA |
| COS | LIST | RES | UNBREAK |
| DATA | LOG | RESEQUENCE | UNTRACE |
| DEF | MAX | RESTORE | UPDATE |
| DELETE | MERGE | RETURN | USING |
| DIGIT | MIN | RND | VAL |
| DIM | NEW | RPT$ | VALIDATE |
| DISPLAY | NEXT | RUN | VARIABLE |
| ELSE | NOT | SAVE | WARNING |
| END | NUM | SEG$ | XOR |

I seguenti sono esempi di nomi validi di variabili:
*Numerici:* X, A9, ALPHA, BASE_PAY, V(3), T(X,Y,Z,Q,A,R,P6), TABLE(Q37,M/4) .
*Stringhe:* S$, YZ2$, NAME$, Q5$(X,7,L/2), ADDRESS$(4)

**ESPRESSIONI NUMERICHE**

Le espressioni numeriche sono formate da costanti numeriche, variabili numeriche e funzioni che usano gli operatori numerici per l'addizione (+), sottrazione (-), moltiplicazione (*), divisione (/), elevazione a potenza ( ).

Il segno meno (-), puo' essere usato sia per indicare una sottrazione che per indicare un numero negativo. Utilizzato come segno di negativo, cambia il segno del valore che lo segue. Per esempio -3 2 e' uguale a -9, lo stesso avviene se viene messo prima della parentesi -(3 2).

La priorita' nell'esecuzione delle operazioni in un'espressione numerica e': elevamento a potenza, moltiplicazione e divisione, addizione e sottrazione. Tuttavia qualsiasi parte di un'espressione numerica che e' compresa tra parentesi, viene eseguita per prima. Qui di seguito viene mostrato l'effetto delle parentesi per determinare il valore di un'espressione:

| *Espressione* | *Risultato* | *Intermedio* | *Finale* |
|---|---|---|---|
| 4+2 2/2-6 | 4+4/2-6 | 4+2-6 | 0 |
| (4+2) 2/2-6 | 6 2/2-6 | 36/2-6 | 12 |
| 4+ 2/(2-6) | 4+4/(-4) | 4-1 | 3 |

**ESPRESSIONI DI STRINGA**

Le espressioni di stringa sono formate da variabili di stringa, costanti di stringa e riferimenti a funzioni che si riferiscono al concatenamento per combinare le stringhe. Se una stringa supera la lunghezza di 255 caratteri, i caratteri in eccedenza sulla destra sono eliminati e viene visualizzato un messaggio di avviso. Il seguente e' un esempio di concatenamento:

```
100 A$="HI" " THERE!"
```

A$="HI" " THERE!" diventa A$ uguale ad "HI THERE!" .

**ESPRESSIONI RELAZIONALI**

Le espressioni relazionali sono spesso usate nell'istruzione IF-THEN-ELSE, ma possono essere usate ovunque siano permesse le espressioni numeriche. Un'espressione relazionale ha il valore di -1 se e' vero il confronto, il valore di 0 se il confronto e' falso. Le espressioni relazionali sono eseguite da sinistra a destra, dopo che tutte le operazioni aritmetiche sono state completate e prima della concatenazione di stringhe (operatore Ampersand). Le espressioni relazionali sono:

| | | | |
|---|---|---|---|
| Uguale a | (=) | Diverso da | (<>) |
| Minore di | (<) | Minore o uguale a | (<=) |
| Maggiore di | (>) | Maggiore o uguale a | (>=) |

I seguenti esempi dimostrano l'uso delle espressioni relazionali:

IF X<Y THEN 200 ELSE GOSUB 420, esegue la linea 200 se X e' minore di Y. Se X e' maggiore o uguale a Y, allora sara' eseguita l'istruzione GOSUB 420.

```
>100 IF X<Y THEN 200 ELSE
 GOSUB 420
```

IF L(C)=12 THEN C=S+1 ELSE CT=CT+1 :: GOTO 140, pone C uguale a S+1 se L(C) e' uguale a 12. Se L(C) non e' uguale a 12, pone CT uguale a CT+1 e dopo viene eseguita la linea 140.

```
>100 IF L(C)=12 THEN C=S+1
 ELSE CT=CT+1 :: GOTO 140
```

A=2<5, pone A uguale a -1 essendo 2 minore di 5.

```
>100 A=2<5
```

PRINT "QUESTO"="QUELLO", stampa 0 poiche' non e' vero che "QUESTO" e' uguale a "QUELLO".

```
>100 PRINT"QUESTO"="QUELLO"
```

A=B=7, pone A uguale a-1 se B e' uguale a 7, o a zero se B e' diverso da 7. Non c'e' alcun effetto su B. Notare che A=B=7 non ha lo stesso significato che ha in aritmetica.

```
>100 A=B=7
```

## ESPRESSIONI LOGICHE

Le espressioni logiche sono usate con le espressioni relazionali. Gli operatori logici sono: AND, OR, NOT e XOR. Se vere, alle espressioni logiche viene attribuito un valore di -1. Se false, ad esse viene dato un valore di 0. L'ordine di precedenza per gli operatori logici e': NOT, XOR, AND e OR.

Un'espressione logica che usa AND e' vera se sono vere le clausole a destra e a sinistra di essa.

Un'espressione logica che usa OR e' vera se sono vere sia la clausola di destra che quella di sinistra, o se lo sono entrambe.

Un'espressione logica che usa NOT e' vera se la clausola che lo segue e' falsa.

Un'espressione logica che usa XOR (OR esclusivo) e' vera sia se sono vere le clausole di sinistra che di destra, ma non se sono entrambe vere.

I seguenti esempi spiegano l'uso delle espressioni logiche:

IF 3<4 AND 5<6 THEN L=7, pone L uguale a 7 poiche' 3 e' minore di 4, e 5 e' minore di 6.

```
>100 IF 3<4 AND 5<6 THEN L=7
```

IF 3<4 AND 5>6 THEN L=7, non pone 7 in L poiche' 3 e' minore di 4, ma 5 non e' maggiore di 6.

```
>100 IF 3<4 AND 5>6 THEN L=7
```

IF 3<4 OR 5>6 THEN L=7, pone 7 in L poiche' 3 e' minore di 4.

```
>100 IF 3<4 OR 5>6 THEN L=7
```

IF 3<4 XOR 5>6 THEN L=7, pone 7 in L poiche' 3 e' minore di 4, e 5 non e' maggiore di 6.

```
>100 IF 3<4 XOR 5>6 THEN L=7
```

IF 3<4 XOR 5<6 THEN L=7, non pone 7 in L perche' 3 e' minore di 4, e 5 e' minore di 6.

```
>100 IF 3<4 XOR 5<6 THEN L=7
```

IF NOT 3=4 THEN L=7, pone 7 in L perche' 3 non e' uguale a 4.

```
>100 IF NOT 3=4 THEN L=7
```

IF NOT 3=4 AND (NOT 6=5 XOR 2=2) THEN 200, non salta alla linea 200 poiche' mentre e' vero che 3 e' diverso da 4, e' anche vero che 6 non e' uguale a 5, e che 2 e' uguale a 2, cosicche' la clausola tra le parentesi risulta falsa.

```
>100 IF NOT 3=4 AND (NOT 6=5
 XOR 2=2) THEN 200
```

IF (A OR B) AND (C XOR D) THEN 200 salta alla linea 200 sia che A o B, che A e B sono veri (uguale a -1), e C o D, ma non C e D sono veri (uguale a -1).

```
>100 IF (A OR B) AND (C XOR
 D) THEN 200
```

Gli operatori logici possono anche essere usati direttamente sui numeri. Essi convertono i numeri in notazione binaria, eseguono l'operazione indicata a livello binario, e poi convertono il risultato nella rappresentazione decimale. Una discussione piu' dettagliata dell'uso degli operatori logici con i numeri si puo' trovare su un testo di matematica o di ingegneria che trattano l'algebra di Boole.

I numeri rappresentati in forma binaria, compresi tra 1000000000000000 a 0111111111111111, devono essere compresi tra -32768 e 32767, con i numeri negativi espressi nella forma del complemento a due con 1 nel posto del bit piu' significativo. Nella notazione binaria, ogni posizione e' una potenza di 2, cosi' come nella notazione decimale e' una potenza di dieci. La tavola seguente mostra la rappresentazione di numeri espressi sia in forma decimale che binaria.

| Posizione decimale | | | Posizione binaria | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 10 | 1 | -16384 | | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

Qui di sopra abbiamo l'equivalente di:
$1_{10}=0000000000000001_2=1_2$ $25_{10}=0000000000011001_2=11001_2$
$6_{10}=0000000000000110_2=110_2$ $-13_{10}=1111111111110011_2$
AND sistema un 1 nella corrispondente posizione binaria se c'e' un 1 sia nel numero che lo precede sia in quello che lo segue. Altrimenti colloca uno 0.
OR mette un 1 nella corrispondente posizione binaria se c'e' un 1 sia nel numero che lo precede sia in quello che lo segue o in entrambi. Altrimenti colloca uno 0.
XOR mette un 1 nella corrispondente posizione binaria se c'e' un 1 sia nel numero che lo precede che in quello che lo segue ma non in entrambi. Altrimenti mette uno 0.
NOT pone un 1 nella corrispondente posizione binaria se c'e' uno 0 nel numero che lo segue. Altrimenti mette uno 0.

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| B: | 2 | 0000000000000010 |
| A AND B: | 0 | 0000000000000000 |
| A: | 6 | 0000000000000110 |
| B: | 5 | 0000000000000101 |
| A AND B: | 4 | 0000000000000100 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| B: | 3 | 0000000000000011 |
| A AND B: | 1 | 0000000000000001 |
| A: | 47 | 0000000000101111 |
| B: | 62 | 0000000000111110 |
| A AND B: | 46 | 0000000000101110 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| B: | 2 | 0000000000000010 |
| A OR B: | 3 | 0000000000000011 |
| A: | 6 | 0000000000000110 |
| B: | 5 | 0000000000000101 |
| A OR B: | 7 | 0000000000000111 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| B: | 3 | 0000000000000011 |
| A OR B: | 3 | 0000000000000011 |
| A: | 47 | 0000000000101111 |
| B: | 62 | 0000000000111110 |
| A OR B: | 63 | 0000000000111111 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| B: | 2 | 0000000000000010 |
| A XOR B: | 3 | 0000000000000011 |
| A: | 6 | 0000000000000110 |
| B: | 5 | 0000000000000101 |
| A XOR B: | 3 | 0000000000000011 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| B: | 3 | 0000000000000011 |
| A XOR B: | 2 | 0000000000000010 |
| A: | 47 | 0000000000101111 |
| B: | 62 | 0000000000111110 |
| A XOR B: | 17 | 0000000000010001 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 1 | 0000000000000001 |
| NOT A: | -2 | 1111111111111110 |
| A: | 6 | 0000000000000110 |
| NOT A: | -7 | 1111111111111001 |

| | DECIMALE | BINARIO |
|---|---|---|
| A: | 2 | 0000000000000010 |
| NOT A: | -3 | 1111111111111101 |
| A: | 47 | 0000000000101111 |
| NOT A: | -48 | 1111111111010000 |

# CAPITOLO 4

## SEZIONE DI RIFERIMENTO

Questo capitolo e' un elenco in ordine alfabetico di tutti i comandi, istruzioni e funzioni del TI Extended BASIC, con una dettagliata spiegazione di come funzionano. Esempi e semplici programmi sono stati inclusi, dove necessario, per maggior chiarezza.

Le parole chiave, poste all'inizio di ogni paragrafo, sono stampate in ENLARGED.
Gli elementi variabili appaiono in *corsivo*. Le parti opzionali sono racchiuse tra parentesi quadre. Le voci che possono essere ripetute sono indicate tra parentesi tonde. Le forme alternative sono rappresentate in colonna una sull'altra.

L'appendice A contiene un elenco dei programmi illustrativi. L'indice da' le pagine dove ciascun elemento del TI Extended BASIC e' usato in un programma illustrativo.

---

# ABS

---

**Formato**

ABS(*espressione numerica*)

**Descrizione**

La funzione . ABS da' il valore assoluto dell'*espressione numerica*. Se l'*espressione numerica* e' positiva, ABS da' il valore dell'espressione numerica. Se l'*espressione numerica* e' negativa, ABS restituisce il suo negativo (un numero positivo). Se l'*espressione numerica* e' 0, ABS restituisce 0. Il risultato di ABS e' sempre un numero non negetivo.

**Esempi**

PRINT ABS(42.3) srampa 42.3 .

```
>100 PRINT ABS(42.3)
```

VV=ABS(-6.124) pone vv uguale a 6.124 .

# ACCEPT

==================================================================

**Formato**

ACCEPT [[AT(*riga, colonna*)][VALIDATE(*tipo di dati,...*)] [BEEP] [ERASE ALL] [SIZE(*espressione numerica*)]:] *variabile*

**Descrizione**

L'istruzione ACCEPT sospende l'esecuzione del programma fino a che non sono stati inseriti i dati dalla tastiera. Con ACCEPT sono disponibili molte opzioni, che lo rendono piu' versatile di INPUT. E' possibile inserire dati da ogni posizione dello schermo, udire un suono (beep) per avvisare che il computer e' pronto ad accettare i dati, cancellare tutti i caratteri sullo schermo prima dell'accettazione limitare ad un certo numero di caratteri i dati da inserire, e limitare il tipo di caratteri da accettare.

**Opzioni**
Le seguenti opzioni possono apparire in un ordine qualsiasi dopo ACCEPT.

AT(*riga, colonna*) pone l'inizio del campo in input a *riga* e *colonna* specificate. Le *righe* sono numerate da 1 a 24. Le *colonne* sono numerate da 1 a 28 con la *colonna* 1 che corrisponde alla *colonna* 3 dei sottoprogrammi VCHAR, HCHAR e GCHAR.

VALIDATE(*tipo di dati,...*) permette l'inserimento solo per certi caratteri. Il *tipo di dati* specifica quali caratteri sono accettabili. Se e' specificato piu' di un *tipo di dati*, e' accettabile un carattere per ciascuno dei tipi di dati specificati. I seguenti sono i *tipi di dati*.

| | |
|---|---|
| UALPHA | consente tutti i caratteri alfabetici maiuscoli. |
| DIGIT | accetta solo le cifre (da 0 a 9). |
| NUMERIC | accetta le cifre da 0 a 9 e i caratteri: ".", "+", "-" ed "E". |

L'*espressione di stringa* consente di immettere i soli caratteri contenuti nell'espressione di stringa.

BEEP emette un breve suono per segnalare che il computer e' pronto ad accettare l'input.

ERASE ALL riempie tutto lo schermo di spazi (blanks) prima di accettare l'input.

SIZE(*espressione numerica*) permette l'immissione del numero di caratteri specificato nell'*espressione numerica* cosi' come segue. Se l'*espressione numerica* e' positiva, prima dell'input viene pulito il campo nel quale il dato in input dovra' essere accettato. Se l'*espressione numerica* e' negativa il campo in input non viene pulito. In questo modo e' possibile fare

accettare un valore di default precedentemente inserito nel campo solo premendo **ENTER**. Se non c'e' la clausola SIZE, la linea viene pulita dal punto di inizio dell'ACCEPT fino alla fine della riga.

Se l'istruzione ACCEPT e' usata senza la clausola AT, gli ultimi due caratteri sullo schermo (in basso a destra) vengono cambiati in caratteri di margine (codice ASCII 31).

**Esempi**

| | |
|---|---|
| ACCEPT AT(5,7):Y accetta il dato alla quinta riga, settima colonna dello schermo e lo pone nella variabile Y. | >100 ACCEPT AT(5,7):Y |
| ACCEPT VALIDATE("YN"):R$ accetta solo i caratteri Y ed N nella variabile R$. | >100 ACCEPT VALIDATE("YN")<br>:R$ |
| ACCEPT ERASE ALL:B accetta il dato nella variabile B dopo aver pulito tutto lo schermo. | >100 ACCEPT ERASE ALL:B |
| ACCEPT AT(R,C)SIZE(LUNCAMPO)BEEP VALIDATE(DIGIT,"AYN"):X$ accetta una cifra o le lettere A, Y o N nella variabile X$, la lunghezza dell'input puo' arrivare fino al numero di caratteri di LUNCAMPO. Il dato e' accettato a riga R, colonna C, ed un *beep* e' emesso prima che il dato sia accettato. | >100 ACCEPT AT(R,C)SIZE(LU<br>NCAMPO)BEEP VALIDATE(DIGI<br>T,"AYN"):X$ |

**Programma**

Il programma sulla destra illustra un tipico uso di ACCEPT. Esso permette di introdurre fino a 20 nomi ed indirizzi e poi li visualizza.

```
>100 DIM NOME$(20),INDR$(2
 0)
>110 DISPLAY AT(5,1)ERASE
 ALL:"NOME:"
>120 DISPLAY AT(7,1):"INDI
 RIZZO:"
>130 DISPLAY AT(23,1):"SCR
 IVI E PER FINIRE"
>140 FOR S=1 TO 20
>150 ACCEPT AT(5,7)VALIDAT
 E(UALPHA,"E")BEEP SIZE(13
 ):NOME$(S)
>160 IF NOME$(S)="E" THEN
 200
>170 ACCEPT AT(7,10)SIZE(1
 2):INDR$(S)
>180 DISPLAY AT(7,10):"
     "
>190 NEXT S
>200 CALL CLEAR
>210 DISPLAY AT(1,1):"NOME
 ","INDIRIZZO"
>220 FOR T=1 TO S-1
>230 DISPLAY AT(T+2,1):NOM
 E$(T),INDR$(T)
>240 NEXT T
>250 GOTO 250
```

(Premere **FCTN 4** per fermare il programma)

---

# ASC

---

### Formato

ASC(*espressione di stringa*)

### Descrizione

La funzione ASC da' il codice ASCII del primo carattere dell'*espressione di stringa*. Una lista dei codici ASCII e' data nell'Appendice C. La funzione ASC e' l'inverso della funzione CHR$.

### Esempio

| | |
|---|---|
| PRINT ASC("A"), stampa 65. | >100 PRINT ASC("A") |
| B=ASC("1"), assegna 49 alla variabile B. | >100 B=ASC("1") |
| DISPLAY ASC("SALVE"), visualizza 83. | >100 DISPLAY ASC("SALVE") |

---

# ATN

---

**Formato**

ATN(*espressione numerica*)

**Descrizione**

La funzione ATN da' la misura dell'angolo (in radianti) la cui tangente e' l'espressione numerica. Se volete l'angolo in gradi, moltiplicatelo per 180/PI. Il valore dato dalla funzione ATN e' sempre compreso nella gamma -PI/2<ATN(X)<PI/2.

**Esempio**

| | |
|---|---|
| PRINT ATN(0), stampa 0. | >100 PRINT ATN(0) |
| Q=ATN(.44), assegna a Q il valore .4145068746. | >100 Q=ATN(.44) |

---

## BREAK

---

**Formato**

BREAK(*numero di linea*)

**Descrizione**

Il comando BREAK richiede un *numero di linea*. Esso fa si che il programma si fermi subito prima che sia eseguita la linea specificata nel *numero di linea*. Dopo che un *breakpoint* e' stato eseguito perche' la riga e' specificata nel *numero di riga* il *breakpoint* viene eliminato e non avvengono piu' interruzioni in quella riga a meno che non sia ordinato da un nuovo comando BREAK o da qualche altra istruzione.

L'istruzione BREAK senza il *numero di linea* fa si che il programma si arresti quando incontra questa istruzione. La linea nella quale il programma si ferma e' chiamata "breakpoint". Ogni volta che si incontra un'istruzione BREAK senza *numero di linea* il programma si ferma anche se e' stata eseguita un'istruzione ON BREAK NEXT.
Potete anche provocare un breakpoint premendo **FCTN 4** mentre il programma e' in esecuzione, a meno che i breakpoints siano stato adoperati in qualche altro modo a causa dell'azione di ON BREAK.

BREAK e' utile per scoprire perche' un programma non stia esattamente eseguendo cio' che ci si aspetta. Quando il programma si e' fermato potete stampare i valori delle variabili per scoprire cosa sta succedendo nel programma. In questo momento e' possibile inserire qualsiasi comando o istruzione che possa essere usata come comando. Se correggete il programma, tuttavia, non si puo' farlo continuare con CON(TINUE).

Un modo per rimuovere una serie di breakpoints con BREAK seguito dai numeri di linea e' il comando UNBREAK. Inoltre, se un breakpoint e' collocato su una linea di un programma e questa viene cancellata, viene rimosso anche il punto d'interruzione. I breakpoints vengono eliminati anche quando si salva un programma con il comando SAVE. Vedi ON BREAK per sapere come trattare i breakpoints.

Ogni qualvolta capita un breakpoint, viene ripristinato il set dei caratteri standard. Cosi' ogni carattere standard che era stato ridefinito con CALL CHAR viene ripristinato. Un breakpoint ripristina anche i colori standard, cancella gli sprites e riporta l'ingrandimento (MAGNIFY), al valore 1 di default.

**Opzioni**

Il *numero di linea* e' facoltativo quando BREAK e' usato come istruzione, ma e' richiesto quando BREAK e' usato come comando.

**Esempi**

| | |
|---|---|
| BREAK come istruzione provoca un breakpoint. | >150 BREAK |
| BREAK 120, 130 come istruzione provoca delle interruzioni prima che avvenga l'esecuzione dei numeri di linea elencati. | >110 BREAK 120,130 |
| BREAK 200,300,1105 come comando provoca delle interruzioni prima dell'escuzione dei numeri di linea elencati. | >BREAK 200,300,1105 |

---

## BYE

---

**Formato**

BYE

**Descrizione**

Con il comando BYE si esce dal TI Extended BASIC e si ritorna alla maschera di accensione del computer. Vengono chiusi tutti i files eventualmente aperti, sono cancellate tutte le linee del programma ed il computer e' resettato. Usare sempre il comando BYE anziche' **FCTN +** (QUIT) per lasciare il TI Extended BASIC. **FCTN +** (QUIT) non chiude i files, il che puo' causare la perdita dei dati conservati su periferiche esterne.

# CALL

==================================================================

## Formato

CALL *sottoprogramma* [(*lista-parametri*)]

## Descrizione

L'istruzione CALL trasferisce il controllo al *sottoprogramma*. Il sottoprogramma puo' essere sia del tipo predefinito, cioe' gia' implementato nel TI Extended BASIC, come ad esempio CLEAR, oppure puo' essere un sottoprogramma scritto da voi. Dopo che il sottoprogramma e' stato eseguito, viene eseguita l'istruzione successiva alla CALL. CALL puo' essere sia un'istruzione o un comando se precede il nome di un sottoprogramma gia' implementato nel linguaggio stesso, ma deve essere un'istruzione quando chiama sottoprogrammi scritti dall'utente.

**Opzioni**

La *lista-parametri* e' definita in funzione del sottoprogramma che state chiamando. Alcuni non richiedono affatto parametri, alcuni li richiedono ed altri hanno parametri facoltativi. In questo manuale ciascun sottoprogramma predefinito e' spiegato piu' avanti sotto la propria voce. La spiegazione dei sottoprogrammi che voi potete scrivere si trova nella sezione del II Capitolo sotto la voce SUB. La seguente e' una lista dei *nomi di sottoprogramma* preesistenti nel TI Extended BASIC.

| | | |
|---|---|---|
| CHAR | HCHAR | PATTERN |
| CHARPAT | INIT | PEEK |
| CHARSET | JOYST | POSITION |
| CLEAR | KEY | SAY |
| COINC | LINK | SCREEN |
| COLOR | LOAD | SOUND |
| DELSPRITE | LOCATE | SPGET |
| DISTANCE | MAGNIFY | SPRITE |
| ERR | MOTION | VCHAR |
| GCHAR | | VERSION |

**Programma**

Il programma sulla destra illustra l'uso di CALL con un sottoprogramma fornito (CLEAR) nella linea 100 e di un sottoprogramma scritto (TIMES) nella linea 120.

```
>100 CALL CLEAR
>110 X=4
>120 CALL TIMES(X)
>130 PRINT X
>140 STOP
>200 SUB TIMES(Z)
>210 Z=Z*P
>220 SUBEND
>RUN
```

sullo schermo appare
12.56637061

## Sottoprogramma CHAR

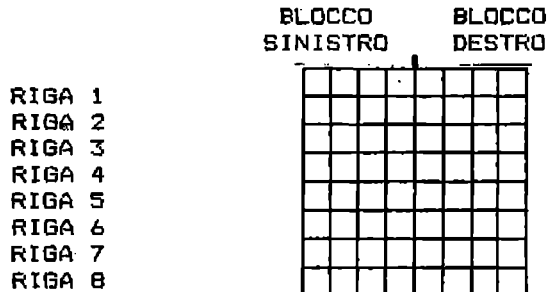
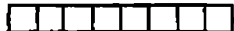
**Formato**

CALL CHAR(*codice-carattere*, *identificatore-sagoma*[,...])

**Descrizione**

Il sottoprogramma CHAR vi permette di definire speciali caratteri grafici. Potete ridefinire il set standard dei caratteri (codici ASCII 32-95) ed i caratteri non definiti (codici ASCII 96-143). *Notare che in TI Extended BASIC sono disponibili meno caratteri definibili che in TI BASIC, dove i caratteri definibili sono compresi fra i codici ASCII 96-156.* Il sottoprogramma CHAR e' l'inverso del sottoprogramma CHARPAT. Il *codice-carattere* specifica qual'e' il carattere che desiderate definire e deve essere un'espressione numerica con un valore compreso tra 32 e 143. L'*identificatore di sagoma* e' un carattere tra 0 e 64 di un'espressione di stringa che specifica la forma del carattere o dei caratteri che volete definire. Questa espressione e' una rappresentazione codificata della matrice di punti che compongono un carattere sullo schermo.
Ciascun carattere e' costituito da 64 puntini (pixels) che formano una griglia di 8x8 come mostrato di seguito.

BLOCCO SINISTRO    BLOCCO DESTRO

RIGA 1
RIGA 2
RIGA 3
RIGA 4
RIGA 5
RIGA 6
RIGA 7
RIGA 8

Ciascuna riga e' suddivisa in 2 blocchi di 4 puntini ciascuno.

Ciascun carattere dell'*identificatore di sagoma* descrive la forma del blocco di una riga. Le righe sono descritte da sinistra a destra e dall'alto in basso. Quindi i primi 2 caratteri dell'*identificatore di sagoma* descrivono la forma della prima riga, i secondi la forma della seconda, e cosi' via.

I caratteri vengono creati accendendo alcuni puntini e lasciandone spenti altri. Il carattere di spazio (codice ASCII 32) non e' altro che un carattere con tutti i puntini spenti. Accendendo tutti i punti si produce un blocco uniforme. Il colore dei puntini accesi e' il colore dei caratteri (foreground). Il colore dei punti spenti corrisponde invece al colore dello sfondo (background).
Tutti i caratteri standard sono rappresentati con gli appropriati punti accesi. Per creare un nuovo carattere, bisogna specificare quali punti vanno accesi e quali rimangono spenti. Nel computer e' utilizzato un codice binario, un numero per ognuno dei 64 puntini (pixels) di ogni singola matrice, per specificare quali punti di un particolare blocco sono accesi o spenti. Una forma piu' vicina alla comprensibilita' umana e' il codice esadecimale. La tabella seguente mostra tutte le possibili condizioni di acceso (1) o spento (0) dei quattro punti di un dato blocco, e i cdici binari ed esadecimali per le relative conbinazioni.

| BLOCCHI | *Codice binario (0=Off 1=On)* | *Codice Esadecimale* |
|---|---|---|
| | 0000 | 0 |
| | 0001 | 1 |
| | 0010 | 2 |
| | 0011 | 3 |
| | 0100 | 4 |
| | 0101 | 5 |
| | 0110 | 6 |
| | 0111 | 7 |
| | 1000 | 8 |
| | 1001 | 9 |
| | 1010 | A |
| | 1011 | B |
| | 1100 | C |
| | 1101 | D |
| | 1110 | E |
| | 1111 | F |

Se l'*identificatore di sagoma* e' minore di 16 caratteri, il computer assegna il valore 0 a quelli mancanti. Se e' da 17 a 32 vengono definiti 2 *codici di caratteri*, il primo fino al 16.o carattere ed il secondo con i caratteri rimanenti, aggiungendo gli zeri se necessario. Per definire 3 caratteri occorrono da 33 a 48 cifre esadecimali. Con 49 e piu' cifre (fino a 64) si ridefiniscono 4 caratteri. Se l'*identificatore di sagoma* ha piu' di 64 caratteri o e' lungo tanto da voler andare a definire oltre il codice ASCII 143, l'accesso e' ignorato.

**Programmi**

Per descrivere la sagoma della matrice disegnata qui appresso, occorre codificare questa stringa con CALL CHAR:

"1898FF3D3C3CE404"

| | BLOCCO SINISTRO | BLOCCO DESTRO | CODICI BLOCCO |
|---|---|---|---|
| RIGA 1 | | | 18 |
| RIGA 2 | | | 98 |
| RIGA 3 | | | FF |
| RIGA 4 | | | 3D |
| RIGA 5 | | | 3C |
| RIGA 6 | | | 3C |
| RIGA 7 | | | E4 |
| RIGA 8 | | | 04 |

Il programma sulla destra usa questa ed un'altra stringa per far sì che una figura si muova.

```
>100 CALL CLEAR
>110 A$="1898FF3D3C3CE404"
>120 B$="1819FFBC3C3C2720"
>130 CALL COLOR(9,7,12)
>140 CALL VCHAR(12,16,96)
>150 CALL CHAR(96,A$)
>160 GOSUB 200
>170 CALL CHAR(96,B$)
>180 GOSUB 200
>190 GOTO 150
>200 FOR PAUSA=1 TO 50
>210 NEXT PAUSA
>220 RETURN
>RUN
```

www.ti99iuc.it

Lo schermo si pulisce
Il carattere si muove
(Premere **FCTN 4** per fermare il programma).

Se un programma si blocca per un'interruzione, i caratteri predefiniti (codici ASCII 32-95) sono riportati alla loro forma standard. Quella con i codici da 96 a 143 mantiene la forma definita nel programma. Quando il programma termina normalmente o a causa di un errore, tutti i caratteri predefiniti vengono resettati.

```
>100 CALL CLEAR
>110 CALL CHAR(96,"FFFFFFFFF
 FFFFFFF")
>120 CALL CHAR(42,"0F0FCF0F0
 F0F0F0F")
>130 CALL HCHAR(12,17,42)
>140 CALL VCHAR(14,17,96)
>150 FOR PAUSA=1 TO 500
>160 NEXT PAUSA
>RUN
```

---

# Sottoprogramma CHARPAT

---

**Formato**

CALL CHARPAT(*codice-carattere,variabile-stringa*[,...])

**Descrizione**

Il sottoprogramma CHARPAT restituisce nella *variabile-stringa* i 16 codici esadecimali dell'identificatore di sagoma che definiscono la sagoma del *codice-carattere*. Il sottoprogramma CHARPAT e' l'inverso del sottoprogramma CHAR. Vedere il sottoprogramma CHAR per una spiegazione del valore riportato nella *variabile-stringa*.

**Esempio**

CALL CHARPAT(33,C$), assegna a C$ la stringa "0010101010001000", che e' l'identificatore di sagoma del carattere 33, il punto esclamativo.

```
>100 CALL CHARPAT(33,C$)
```

## Sottoprogramma CHARSET

**Formato**

CALL CHARSET

**Descrizione**

Il sottoprogramma CHARSET ripristina la sagoma ed i colori standard dei caratteri da 32 a 95. Generalmente quando un programma viene lanciato da un altro programma usando RUN come istruzione, i caratteri compresi tra 32 e 95 non vengono ripristinati alle loro forme e colori standard. CHARSET risulta quindi utile quando questa caratteristica non e' desiderata.

**Esempio**

CALL CHARSET ripristina i colori e la forma standard dei caratteri.

```
>100 CALL CHARSET
```

## CHR$

**Formato**

CHR$(*espressione-numerica*)

**Descrizione**

La funzione CHR$ restituisce il carattere corrispondente al codice del carattere ASCII specificato nell'*espressione-numerica*. La funzione CHR$ e' l'inverso della funzione ASC. Un elenco dei codici dei caratteri ASCII si trova nell'Appendice C.

**Esempi**

PRINT CHR$(72), stampa H.

```
>100 PRINT CHR$(72)
```

X$=CHR$(33), pone X$ uguale al !.

```
>100 X$=CHR$(33)
```

**Programma**

Per avere un elenco completo di tutti i codici ASCII e i loro corrispondenti valori ASCII, eseguire il programma sulla destra.

```
>100 CALL CLEAR
>110 FOR A=32 TO 95
>120 PRINT A;"""";CHR$(A);"""
 ";
>130 NEXT A
```

---

# Sottoprogramma CLEAR

---

**Formato**

CALL CLEAR

**Descrizione**

Il sottoprogramma CLEAR e' usato per pulire lo schermo. Quando e' chiamato il sottoprogramma CLEAR, il carattere di spazio (codice ASCII 32) e' posto in tutte le posizioni dello schermo.

**Programmi**

Quando il programma sulla destra viene lanciato, viene pulito lo schermo prima che siano eseguite le istruzioni PRINT.

```
>100 CALL CLEAR
>110 PRINT "CIAO CARO !"
>120 PRINT "COME STAI ?"
>RUN
Lo schermo si pulisce
CIAO CARO !
COME STAI ?
```

Se il carattere di spazio (codice ASCII 32) e' stato ridefinito con il sottoprogramma CALL CHAR, lo schermo si riempie con i nuovi caratteri quando viene eseguito CALL CLEAR.

```
>100 CALL CHAR(32,"0103070F1F
 3F7FFF")
>110 CALL CLEAR
>120 GOTO 120
>RUN
lo schermo si riempie con ◢
(premere FCTN 4 per fermare
il programma).
```

---

# CLOSE

---

**Formato**

CLOSE *#numero-file*[:DELETE]

**Descrizione**

L'istruzione CLOSE ferma l'uso del file di un programma a cui si fa riferimento nel *#numero-file*. Dopo che e' stata eseguita l'istruzione CLOSE, il file non puo' essere usato nel programma a meno che non sia di nuovo aperto con OPEN. Il computer non associa piu' il *#numero-file* con il file chiuso, cosicche' e' possibile assegnare a quel numero un altro file.

Quando nessun programma e' in esecuzione le seguenti azioni chiudono tutti i files aperti:

Correzione del programma (modo Edit)
Inserimento del comando BYE
Inserimento del comando RUN
Inserimento del comando NEW
Inserimento del comando OLD
Inserimento del comando SAVE
Inserimento del comando LIST ad una periferica

Se usate **FCTN** + (QUIT) per lasciare il TI Extended BASIC, il computer non chiude nessun file precedentemente aperto, e si possono perdere i dati su ogni file eventualmente aperto. Per evitare cio', e' opportuno lasciare il TI Extended BASIC con il comando BYE anziche' **FCTN** + (QUIT).

**Opzioni**

Potete cancellare un file dal disco nello stesso momento che lo chiudete, aggiungendo ":DELETE" all'istruzione. Altre periferiche, come i registratori a cassette, non permettono l'uso di DELETE. Il manuale di ciascuna periferica spiega l'uso di DELETE.

**Esempi**

Quando il computer esegue l'istruzione CLOSE usando un registratore a cassette, compaiono sullo schermo le istruzioni per manovrare il registratore.

```
>100 OPEN #24:"CS1",INTERNAL,
 INPUT,FIXED
 -
 -
 -
 linee di programma
 -
 -
 -
>200 CLOSE #24
>RUN
 istruzioni di apertura
 -
 -
 -
 il programma gira
 -
 -
 -
 * PRESS CASSETTE STOP       CS1
   THEN PRESS ENTER
```

L'istruzione CLOSE per un dischetto non rihiede nessuna altra azione da parte vostra.

```
>100 OPEN #24:"DSK1.nome-file",
 INTERNAL,INPUT,FIXED
 -
 -
 -
 linee programma
 -
 -
 -
>200 CLOSE #24
>RUN
 il programma gira
```

---

# Sottoprogramma COINC

---

**Formato**

CALL COINC(#*num.-sprite,num.-sprite,tolleranza,variab.-numerica*)
CALL COINC(#*num.-sprite,pixel-riga,pixel-colonna,tolleranza,*
*variab.-numerica*)
CALL COINC(ALL,*variabile-numerica*)

**Descrizione**

Il sottoprogramma COINC verifica se c'e' coincidenza tra due sprites o tra uno sprite e una posizione sullo schermo.
Il valore riportato nella *variabile-numerica*, e' -1 se c'e' una coincidenza e 0 se non c'e' coincidenza.

Se e' usata la parola chiave ALL viene riportata la coincidenza tra ogni coppia di sprites. Se due sprites sono identificati da un *#numero-sprite*, viene riportata la loro coincidenza. Se sono identificati un *#numero-sprite* ed una locazione dello schermo, viene riportata la loro coincidenza.

Se viene usata la parola chiave ALL, gli sprites sono coincidenti solo se uno o piu' punti (pixels) di quelli che li costituiscono, occupano contemporaneamente la stessa posizione sullo schermo. Se sono dati due sprites o uno sprite ed una posizione, allora deve essere specificata la *tolleranza* e due sprites sono coincidenti se i loro due angoli superiori sinistri sono compresi nel valore specificato nella *tolleranza*. Uno sprite ed una posizione sono coincidenti se l'angolo superiore sinistro dello sprite e la posizione specificata nel *pixel di riga* e nel *pixel di colonna* sono compresi nel valore specificato dalla *tolleranza*. Queste coincidenze sono riportate anche se non c'e' nessuna apparente sovrapposizione tra gli sprites o tra uno sprite ed una posizione.

*Pixel-riga* e *pixel-colonna* sono numerati consecutivamente partendo da 1 a cominciare dall'angolo superiore sinistro dello schermo. Cosi' il *pixel-riga* puo' essere compreso tra 1 e 192, ed il *pixel-colonna* da 1 a 256. (In realta' il *pixel-riga* puo' arrivare fino a 256 ma le posizioni da 193 a 256 sono fuori della parte inferiore dello schermo. Se ogni parte dello sprite occupa la posizione data allora c'e' una coincidenza.

Il verificarsi o meno di una coincidenza dipende da diversi fattori. Se gli sprites si stanno muovendo molto velocemente, COINC non puo' essere in grado di verificare la loro coincidenza. Inoltre COINC verifica una coincidenza solo quando esso e' chiamato, cosi' un programma puo' lasciarsi sfuggire una coincidenza quando il programma sta eseguendo qualche altra istruzione.

**Programma**

Il programma sulla destra definisce due sprites che consistono in un triangolo.

```
>100 CALL CLEAR
>110 S$="0103070F1F3F7FFF"
>120 CALL CHAR(96,S$)
>130 CALL CHAR(100,S$)
>140 CALL SPRITE(#1,96,7,8,8)
>150 CALL SPRITE(#2,100,5,1,1)
```

La linea 160 mostra una coincidenza perche' gli sprites sono per 10 pixels uno all'interno dell'altro.

```
>160 CALL COINC(#1,#2,10,C)
>170 PRINT C
```

La linea 180 non mostra coincidenze perche' le aree disegnate degli sprites non sono coincidenti.

```
>180 CALL COINC(ALL,C)
>190 PRINT C
>RUN
 -1
  0
```

---

# Sottoprogramma COLOR

===

## Formato

```
CALL COLOR(#numero-sprite,colore-sprite[,...])
CALL COLOR(set-caratteri,col-carattere,col-sfondo[,...])
```

## Descrizione

Il sottoprogramma COLOR permette di specificare sia il *colore-sprite* relativo al *#numero-sprite* o un *colore-carattere* ed un *colore-sfondo* per i caratteri specificati nel *set-caratteri*. In un determinato CALL COLOR e' possibile definire il colore o i colori degli sprites, o i colori del set dei caratteri, ma non entrambi.

Ciascun carattere ha due colori. Il colore dei pixels che costituiscono il carattere stesso e' chiamato *foreground-color*. Il colore che occupa il resto della posizione che il carattere occupa sullo schermo e' chiamato *background-color*. Negli sprites il colore di sfondo, *background-color* e' sempre il codice 1 (trasparente), che ne permette la visibilita' sullo schermo qualunque ne sia il colore. Per cambiare il colore dello schermo vedere il sottoprogramma SCREEN.
*Foreground-color* e *background-color* devono avere valori compresi tra 1 e 16. I codici dei colori sono descritti qui appresso:

| *Codice Colore* | *Colore* |
|---|---|
| 1 | Trasparente |
| 2 | Nero |
| 3 | Verde |
| 4 | Verde Chiaro |
| 5 | Blu Scuro |
| 6 | Blu Chiaro |
| 7 | Rosso Scuro |
| 8 | Ciano |
| 9 | Rosso |
| 10 | Rosso Chiaro |
| 11 | Giallo Scuro |
| 12 | Giallo Chiaro |
| 13 | Verde Scuro |
| 14 | Magenta |
| 15 | Grigio |
| 16 | Bianco |

Finche' CALL COLOR non viene eseguito, lo standard *foreground-color* e' nero (codice 2) e il *background-color* di sfondo e' trasparente (codice 1) per tutti i caratteri. Il colore viene assegnato agli sprites quando questi vengono creati. Quando

avviene un'interruzione, tutti i caratteri vengono riportati ai loro colori standard.
Per usare CALL COLOR occorre anche specificare a quale dei 15 sets il carattere appartiene. (*Notare che il TI BASIC ha 16 sets mentre il TI Extended BASIC ne ha 15.*) La lista dei codici dei caratteri ASCII per i caratteri standard si trova nell'Appendice C. I numeri dei *sets dei caratteri* sono dati qui di seguito:

| *Numero set* | *Codici caratteri* |
|---|---|
| 0 | 30-31 |
| 1 | 32-39 |
| 2 | 40-47 |
| 3 | 48-55 |
| 4 | 56-63 |
| 5 | 64-71 |
| 6 | 72-79 |
| 7 | 80-87 |
| 8 | 88-95 |
| 9 | 96-103 |
| 10 | 104-111 |
| 11 | 112-119 |
| 12 | 120-127 |
| 13 | 128-135 |
| 14 | 136-143 |

**Esempi**

CALL COLOR(3,5,8) stabilisce il *foreground color* dei caratteri da 48-55 a 5 (blu scuro) ed il *background color* a 8 (ciano).

```
>100 CALL COLOR(3,5,8)
```

CALL COLOR(#5,16) stabilisce che lo sprite n.5 abbia un colore (*foreground*) di 16 (bianco). Il *background color* e' sempre 1 (trasparente).

```
>100 CALL COLOR(#5,16)
```

CALL COLOR(#7,INT(RND+1)) stabilisce che lo sprite n.7 abbia un colore scelto casualmente tra i 16 colori disponibili.

```
>100 CALL COLOR(#7,INT(RND
  +1))
```

---

# CONTINUE

---

**Formato**

CONTINUE
CON

**Descrizione**

Il comando CONTINUE fa riprendere l'esecuzione di un programma fermato da un breakpoint. Esso puo' essere inserito ogni volta che un programma si interrompe a causa di un breakpoint provocato dal comando BREAK o dall'istruzione FCTN 4 (CLEAR). Tuttavia non si puo' usare il comando CONTINUE se e' stata corretta una linea di programma. CONTINUE puo' essere abbreviato in CON.

Quando avviene un breakpoint sono ripristinati i caratteri e i colori standard. Gli sprites cessano di esistere. CONTINUE non ripristina ne' i colori e ne' i caratteri standard che sono stati eventualmente ridefiniti. Altrimenti il programma continua normalmente se non e' avvenuta nessuna interruzione.

---

# COS

---

### Formato

COS(*espressione in radianti*)

### Descrizione

La funzione COS da' il coseno trigonometrico dell'*espressione in radianti*. Se l'angolo e' dato in gradi moltiplicare il numero dei gradi per PI/180 per avere l'equivalente angolo in radianti.

### Programma

Il programma sulla destra fornisce il coseno di vari angoli.

```
>100 A=1.047197551196
>110 B=60
>120 C=45*PI/180
>130 PRINT COS(A);COS(B)
>140 PRINT COS(B*PI/180)
>150 PRINT COS(C)
>RUN
.5 -.9524129804
.5
.7071067812
```

---

# DATA

---

**Formato**

DATA *lista dati*

**Descrizione**

L'istruzione DATA permette di conservare dati all'interno del vostro programma. I dati, che possono essere numerici o alfabetici, sono elencati nella *lista dati* separati da una virgola. Durante l'esecuzione di un programma, l'istruzione READ assegna i valori della *lista dati* alle variabili specificate nella *lista variabili* dell'istruzione READ.

Le istruzioni DATA possono essere poste in qualsiasi parte del programma. Quello che e' importante e' l'ordine con le quali esse compaiono. I valori delle istruzioni DATA sono letti sequenzialmente, a cominciare dal priomo valore della prima istruzione DATA che compare nel programma. Se un programma ha piu' di un'istruzione DATA le istruzioni DATA vengono lette nell'ordine con il quale esse compaiono nel programma, a meno che non sia stato altrimenti specificato da un'istruzione RESTORE. Cosi' l'ordine con il quale i DATA compaiono nel programma determina in genere l'ordine con il quale i dati sono letti. I DATA non possono far parte di una linea con piu' istruzioni.

I dati nella *lista dati* devono corrispondere al tipo di variabile assegnato nell'istruzione READ. Cosi' se e' specificata una variabile numerica nell'istruzione READ, una costante numerica deve trovarsi nella corrispondente posizione dell'istruzione DATA. Allo stesso modo ad una variabile alfanumerica deve corrispondere una stringa. Un numero in un'istruzione DATA e' considerato stringa valida se letto mediante un'istruzione READ con una variabile stringa. Se un'istruzione DATA contiene virgole vicine, il computer interpreta che vogliate inserire una stringa nulla (una stringa con nessun carattere).

Quando si usano costanti di stringa in un'istruzione DATA, si possono includere le stringhe tra virgolette. Tuttavia, se la stringa inclusa contiene una virgola, spazi all'inizio o alla fine, dovete chiudere la stringa tra virgolette. Se la stringa e' racchiusa tra virgolette, questi apici vengono rappresentati da apici doppi.

**Programma**

Il programma sulla destra legge e stampa alcune costanti numeriche e di stringa. Le linee da 100 a 130 leggono 5 gruppi di dati e stampano i loro valori, due per riga.

```
>100 FOR A=1 TO 5
>110 READ B,C
>120 PRINT B;C
>130 NEXT A
>140 DATA 2,4,6,7,8
>150 DATA 1,2,3,4,5
>160 DATA """QUESTA HA APICI"
 ""
>170 DATA " SENZA APICI, QUI"
>180 DATA ANCHE QUI SENZA API
 CI
```

Le linee da 190 a 220 leggono 7 elementi di dati e ne stampano ognuno sulla propria riga.

```
>190 FOR A=1 TO 7
>200 READ B$
>210 PRINT B$
>220 NEXT A
>230 DATA 1,NUMERO,,TI
>RUN
```

| | |
|---|---|
| I primi 2 elementi della linea 140. | 2 4 |
| I secondi 2 elementi della linea 140. | 6 7 |
| Ultimo elemento della linea 140 e primo della linea 150. | 8 1 |
| Secondo e terzo elemento della linea 150. | 2 3 |
| Quarto e quinto elemento della linea 150. | 4 5 |
| Linea 160. | "QUESTA HA APICI" |
| Linea 170. | SENZA APICI, QUI |
| Linea 180. | ANCHE QUI SENZA APICI |
| Primo elemento della linea 230. | 1 |
| Secondo elemento della linea 230. | NUMERO |
| Stringa nulla delle 2 virgole nella linea 230. | |
| Ultimo elemento della linea 230. | TI |

---

# DEF

===

**Formato**

DEF *nome-funzione*[(*parametro*)] =*espressione*

**Descrizione**
L'istruzione DEF vi permette di definire le vostre funzioni. Il *nome-funzione* puo' essere qualsiasi nome di variabile. Se specificate un *parametro*, esso deve essere racchiuso tra parentesi e puo' essere un qualsiasi nome di variabile scalare. Se l'*espressione* e' una stringa, il *nome-funzione* deve essere il nome di una variabile di stringa, cioe' l'ultimo carattere deve essere il segno $.

L'istruzione DEF deve trovarsi ad un numero di linea piu' basso di quello in cui e' usata la funzione. Tuttavia un'istruzione DEF non puo' comparire in un'istruzione IF-THEN-ELSE. Quando il computer incontra un'istruzione DEF, esso procede all'istruzione successiva senza compiere alcuna azione. Una funzione puo' essere usata in qualsiasi espressione numerica o di stringa usando il *nome-funzione* seguito da un'espressione racchiusa tra parentesi se un *parametro* era stato specificato nell'istruzione DEF.

Quando in un'espressione si incontra un rimando alla funzione (usando il *nome-funzione* in un'istruzione), la funzione e' espressa secondo i valori correnti delle variabili specificate nell'istruzione DEF ed il valore del *parametro* se esso esiste. Un'istruzione DEF puo' riferirsi ad altre funzioni definite. Tuttavia, la funzione specificata non puo' riferirsi a se stessa ne' direttamente (per es. DEF B=B*2) ne' indirettamente (per es. DEF F=G::DEF G=F).

Il tentativo di stampare il valore di una funzione con PRINT usato come comando non ha alcun effetto se l'Espansione di Memoria e' collegata al computer.

**Opzioni**

Se specificate un *parametro* per una funzione, quando si incontra in un'espressione un rimando alla funzione, il suo valore viene assegnato al *parametro*. Il valore della funzione viene quindi determinato usando il valore del *parametro* ed i valori delle altre variabili che compaiono nell'istruzioni DEF. Se in un'istruzione DEF viene fornito il *parametro*, deve essere sempre dato un valore all'argomento quando ci si riferisce alla funzione.

Il nome *parametro* usato nell'istruzione DEF influenza solamente l'istruzione DEF in cui esso e' usato. Questo significa che esso e' distinto da ogni altra variabile con lo stesso nome che puo' apparire in altra parte del programma.

Il *parametro* non puo' essere usato come un array. Potete usare

un elemento dell'array in una funzione finche' l'array non ha lo stesso nome del *parametro*. Per es. potete usare DEF F(A) ma non DEF F(A)=A(Z).

**Esempi**

DEF PAY(OT)=40*RATE+1.5*RATE*OT definisce PAY cosi' che ogni volta che e' incontrato in un programma la variabile PAY e' raffigurata usando il risultato dell'operazione.

```
>100 DEF PAY(OT)=40*RATE+1.5*
 RATE*OT
```

DEF RND20=INT(RND+1) definisce RND20 in modo che ogni volta che viene incontrato nel programma viene dato a caso un numero intero compreso tra 1 e 20.

```
>100 DEF RND20=INT(RND+1)
```

DEF PAROLA1$(NOME$)=SEG$(NOME$,1, POS(NOME$," ",1)-1) definisce PAROLA1$ come la parte di NOME$ che precede uno spazio.

```
>100 DEF PAROLA1$(NOME$)=SEG$
 (NOME$,1,POS(NOME$," ",1)-1)
```

---

# DELETE

---

**Formato**

DELETE *periferica-nome file*

**descrizione**

Il comando DELETE vi permette di rimuovere un programma o un file da un dispositivo di memoria di massa del computer. La *periferica-nome file* deve essere un'espressione di stringa. Se si usa una costante di stringa, questa deve essere inclusa tra apici. Si possono anche cancellare files di dati usando la parola DELETE nell'istruzione CLOSE.

Alcune periferiche (come i dischi) permettono la cancellazione di files; altri (come le cassette) no. Riferirsi ai singoli manuali per maggiori informazioni.

**Esempio**

DELETE "DSK1.MYFILE" cancella il file chiamato MYFILE dal dischetto nel Disk Drive 1.

```
>DELETE "DSK1.MYFILE"
```

**Programma**

Il programma sulla destra dimostra un uso di DELETE.

```
>100 INPUT "FILENAME: ":X$
>110 DELETE X$
```

---

# Sottoprogramma DELSPRITE

===

**Formato**

CALL DELSPRITE(#*Numero-sprite*[,...]) CALL DELSPRITE(ALL)

**Descrizione**

Il sottoprogramma DELSPRITE cancella gli sprites dallo schermo. Si possono cancellare uno o piu' sprites specificando il loro numero preceduto dal segno # e separadoli con virgole, oppure si possono cancellare tutti gli sprites esistenti mediante la spcifica ALL. Dopo essere stato cancellato con DELSPRITE, uno sprite puo' essere ricreato mediante il sottoprogramma SPRITE.

**Esempi**

CALL DELSPRITE(#3) cancella lo sprite numero 3.

```
>100 CALL DELSPRITE(#3)
```

CALL DELSPRITE(#4,#3*C) cancella lo sprite numero 4 e lo sprite il cui numero e' il risultato della moltiplicazione di C per 3.

```
>100 CALL DELSPRITE(#4,#3*C)
```

CALL DELSPRITE(ALL) cancella tutti gli sprites.

```
>100 CALL DELSPRITE(ALL)
```

# DIM

### Formato

DIM *nome-matrice*(*intero1* [,*intero2*]...[,*intero7*])[,...]

### Descrizione

L'istruzione DIM riserva spazio nella memoria del computer per le matrici numeriche e di stringa. Potete dimensionare una matrice una sola volta in un programma. Se dimensionate una matrice, l'istruzione DIM deve comparire nel programma alla linea con la numerazione piu' bassa rispetto alle altre linee che fanno riferimento alla matrice. Se dimensionate piu' di una matrice in una sola istruzione DIM, i *nomi-matrici* devono essere separati da virgole. Il *nome-matrice* puo' essere un qualsiasi nome di variabile. Un'istruzione DIM non puo' comparire in un'istruzione IF-THEN-ELSE.
Con il TI Extended BASIC si possono dimensionare le variabili fino a 7 indici. I numeri *interi* separati da virgola, seguenti al *nome-matrice* determinano quante dimensioni ha la matrice. I valori degli *interi* determinano il numero degli elementi di ciascuna dimensione.
Lo spazio per una matrice viene assegnato dopo aver digitato il comando RUN ma prima che siano eseguite le altre istruzioni. Ciascun elemento di una matrice alfabetica e' una stringa nulla e ciascun elemento di una matrice numerica e' 0 finche' non vengano loro assegnati i valori corrispondenti. I valori degli interi determinano il massimo valore di ciascun indice di quella matrice. Se usate una matrice non definita in un'istruzione DIM il massimo valore di ciascun indice e' 10. Il primo elemento e' 0 a meno che un'istruzione OPTION BASE non assegni al primo indice il valore 1. Cosi' una matrice definita come DIM A(6) e' una matrice dimensionata con 7 elementi, a meno che l'indice 0 non sia stato eliminato dall'istruzione OPTION BASE.

### Esempi

| | |
|---|---|
| DIM X$(30) riserva spazio nella memoria del computer per 31 elementi della matrice chiamata X$. | >100 DIM X$(30) |
| DIM D(100),B(10,9) riserva spazio in memoria per 101 elementi della matrice chiamata D e 110 (11x10) elementi della matrice chiamata B. | >100 DIM D(100),B(10,9) |

---

# DISPLAY

---

**Formato**

DISPLAY [[AT(*riga,colonna*)][BEEP][ERASE ALL][SIZE(*espressione-numerica*)]:]*lista-stampa*

**Descrizione**

L'istruzione DISPLAY visualizza informazioni sullo schermo. Sono disponibili varie opzioni, che rendono DISPLAY molto piu' versatile di PRINT. E' possibile visualizzare dati in ogni posizione dello schermo, e nello stesso tempo far emettere un suono acustico al computer, pulire posizioni dello schermo, e cancellare tutti i caratteri prima di visualizzare i dati.

**Opzioni**

AT(*riga,colonna*) mette l'inizio del campo di visualizzazione dei dati alla *riga* e *colonna* specificate. Le righe sono numerate da 1 a 24. Le colonne sono numerate da 1 a 28, con la colonna 1 corrispondente alla colonna 3 dei sottoprogrammi VCHAR, HCHAR e GCHAR. Se non e' presente l'opzione AT il dato viene visualizzato a riga 24, colonna 1, esattamente come con l'istruzione PRINT.
BEEP emette un breve suono quando viene visualizzato il dato.
ERASE ALL riempie tutto lo schermo con i caratteri di spazio (32) prima di visualizzare il dato.
SIZE(*espressione-numerica*) assegna la lunghezza dei caratteri da visualizzare a partire dalla riga e colonna assegnate. Se non utilizzate tutta la riga questa e' completata con gli spazi. Se l'*espressione-numerica* e' maggiore del numero di posizioni restanti nella riga, viene pulito solo il resto della riga.

**Esempi**

DISPLAY AT(5,7):Y visualizza il valore di Y a riga 5 colonna 7 dello schermo.

```
>100 DISPLAY AT(5,7):Y
```

DISPLAY ERASE ALL:B pulisce tutto lo schermo prima di visualizzare il valore di B.

```
>100 DISPLAY ERASE ALL:B
```

DISPLAY AT(R,C)SIZE(LUNGCAMPO) BEEP:X$ visualizza il valore di X$ a riga R, colonna C. Subito prima emette un BEEP e pulisce il numero di caratteri specificati in LUNGCAMPO.

```
>100 DISPLAY AT(R,C) SIZE(LUN
 GCAMPO)BEEP:X$
```

**Programma**

Il programma sulla destra e' una dimostrazione dell'uso di DISPLAY. Esso permette di piazzare blocchi in qualsiasi posizione dello schermo per tracciare una figura o disegnare.

```
>100 CALL CLEAR
>110 CALL COLOR(9,5,5)
>120 DISPLAY AT(23,1):"INTROD
 UCI RIGA E COLONNA."
>130 DISPLAY AT(24,1):"RIGA:
   COLONNA:"
>140 FOR CONT=1 TO 2
>150 CALL KEY(0,RIGA(CONT),S)
>160 IF S<=0 THEN 150
>170 DISPLAY AT(24,5+CONT) SI
 ZE(1):STR$(RIGA(CONT)-48)
>180 NEXT CONT
>190 FOR CONT=1 TO 2
>200 CALL KEY(0,COL(CONT),S)
>210 IF S<=0 THEN 200
>220 DISPLAY AT(24,16+CONT) S
 IZE(1):STR$(COL(CONT)-48)
>230 NEXT CONT
>240 RIGA1=10*(RIGA(1)-48)*RI
 GA(2)-48
>250 COL1=10*(COL(1)-48)+COL(
 2)-48
>260 DISPLAY AT(RIGA1,COL1) S
 IZE(1):CHR$(96)
>270 GOTO 130
```

(Premere **FCTN 4** per fermare il programma)

---

# DISPLAY USING

---

**Formato**

DISPLAY [*lista-opzioni*:] USING *espressione-stringa* [:*lista di stampa*]
DISPLAY [*lista-opzioni*:] USING *numero-linea* [:*lista di stampa*]

**Descrizione**

L'istruzione DISPLAY...USING e' uguale alla DISPLAY con l'aggiunta della clausola USING, la quale specifica il formato dei dati elencati nella *lista di stampa*. Se e' presente l'*espressione-stringa*, essa definisce il formato. Se e' presente il *numero-linea*, questo si riferisce al numero di linea di un'istruzione IMAGE. Vedere IMAGE per una spiegazione sull'uso di questo formato.

**Esempi**

DISPLAY AT(10,4):USING "##.##":N visualizza il valore di N a riga 10 e colonna 4 con il formato "##.##" .

```
>100 DISPLAY AT(10,4):USING
 "##.##":N
```

DISPLAY USING "##.##":N visualizza il valore di N a riga 24 colonna 1 con il formato "##.##" .

```
>100 DISPLAY USING "##.##":N
```

Sono valide anche le seguenti istruzioni:

```
>100 PRINT USING A$:X,Y

>100 DISPLAY USING RPT$("#",5
 )&V$:A(12)
```

---

# Sottoprogramma DISTANCE

---

**Formato**

CALL DISTANCE(*#numero-sprite,#numero-sprite,variabile-numerica*)
CALL DISTANCE(*#numero-sprite,punto-riga,punto-colonna,variabile-numerica*)

**Descrizione**

Il sottoprogramma DISTANCE da' il quadrato della distanza tra 2 sprites o tra uno sprite ed una locazione dello schermo. La posizione di ciascuno sprite e' considerata a partire dall'angolo superiore sinistro. *Punto-riga* e *punto-colonna* vanno da 1 a 256. La distanza al quadrato e' riportata nella *variabile-numerica*.

Il numero dato e' calcolato come segue: la differenza tra i pixels di riga dei due sprites (o dello sprite e della locazione) viene calcolata e riportata al quadrato. Poi la differenza tra i pixels di colonna dei due sprites (o dello sprite e della locazione) e' calcolata e riportata al quadrato. Poi i due quadrati vengono sommati. Se la somma e' maggiore di 32767 viene riportato 32767. La distanza tra gli sprites (o tra lo sprite e la locazione) e' la radice quadrata del valore calcolato.

**Esempi**

CALL DISTANCE(#3,#4,DIST) pone DIST uguale al quadrato della distanza tra gli angoli superiori sinistri degli sprites #3 e #4.

```
>100 CALL DISTANCE(#3,#4,DIST)
```

CALL DISTANCE(#4,18,89,D) pone D uguale al quadrato della distanza tra l'angolo superiore sinistro dello sprite #4 e la posizione 18, 89 dello schermo.

```
>100 CALL DISTANCE(#4,18,89,D)
```

---

# END

---

**Formato**

END

**Descrizione**

L'istruzione END pone termine al programma e puo' essere usata in alternativa all'istruzione STOP. Sebbene l'istruzione END possa apparire ovunque, essa e' normalmente posta all'ultima linea di un programma, definendo in questo modo la sua fine logica e fisica. L'istruzione STOP e' usata in altri posti dove si desidera che il programma si arresti. In TI Extended BASIC non e' obbligatorio usare l'istruzione END. Il programma termina automaticamente dopo aver eseguito l'ultima linea.

---

## EOF (End-Of-File)

---

**Formato**

EOF(*numero-file*)

**Descrizione**

La funzione EOF e' utilizzata per verificare se ci sono altri records da leggere in un file. Il valore del *numero-file* indica il file e deve corrispondere al numero di un file aperto. La funzione EOF non puo' essere usata con le cassette.

La funzione EOF presuppone sempre che il prossimo record sara' letto sequenzialmente anche se si sta' usando un file RELATIVE.

Il valore che la funzione EOF fornisce dipende dalla posizione del file. Se non si e' all'ultimo record, la funzione restituisce il valore 0. Se si e' all'ultimo record, la funzione riporta il valore 1. Se il dischetto o un altro dispositivo di memoria di massa e' pieno si e' alla fine fisica del file, poiche' non c'e' piu' spazio per i dati; la funzione riporta un valore di -1.

Per maggiori informazioni riferirsi al manuale del sistema di memoria a dischi.

**Esempi**

PRINT EOF(3) stampa un valore a seconda se si e' alla fine del file che era stato aperto con #3.

```
>100 PRINT EOF(3)
```

IF EOF(27)<>0 THEN 1150 trasferisce il controllo alla linea 1150 se si e' alla fine del file che era stato aperto con #27.

```
>100 IF EOF(27)<>0 THEN 1150
```

IF EOF(27) THEN 1150 trasferisce il controllo alla linea 1150 se si e' alla fine del file che era stato aperto con #27.

```
>100 IF EOF(27) THEN 1150
```

---

# Sottoprogramma ERR

---

**Formato**

CALL ERR(*codice-errore,tipo-errore* [,*gravita'-errore,*
*numero-linea*])

**Descrizione**

Il sottoprogramma ERR riporta il *codice-errore* ed il *tipo-errore* dell'ultimo errore avvenuto. Un errore viene rimosso quando e' stato ammesso dal sottoprogramma ERR, quando avviene un altro errore o quando il programma e' finito.

I *codici-errore* sono numeri di due o tre cifre. Il significato di ciascun codice si trova nell'Appendice N.

Se il *tipo-errore* e' un numero negativo, l'errore era allora nell'esecuzione del programma. Se il *codice-errore* e' 130 (I/O ERROR), il *tipo-errore* e' un numero positivo ed il numero e' il numero del file che ha causato l'errore.

Se non si sono verificati errori, CALL ERR restituisce tutti valori 0.

CALL ERR e' usato in congiunzione con ON ERROR.

**Opzioni**

E' possibile ottenere la *gravita'-errore* ed il *numero-linea* di dove e' avvenuto l'errore. La *gravita'-errore* e' sempre 9. Il *numero-linea* e' il numero della linea in esecuzione al momento dell'errore. Non sempre la linea specificata e' l'origine del problema poiche' un errore puo' capitare a causa di valori generati o azioni eseguite in qualche altra parte del programma.

**Esempi**

| | |
|---|---|
| CALL ERR(A,B) pone A uguale al *codice-errore* e B uguale al *tipo-errore* dell'ultimo errore. | >100 CALL ERR(A,B) |
| CALL ERR(W,X,Y,Z) pone W uguale al *codice-errore*, X uguale al *tipo-errore*, Y uguale alla *gravita'-errore* e Z uguale al *numero-linea* dell'ultimo errore. | >100 CALL ERR(W,X,Y,Z) |

**Programma**

Il programma sulla destra dimostra l'uso di CALL ERR. Un errore viene provocato nella linea 110 a causa dell'illegale codice di colore dello schermo. Per azione della linea 100, il controllo e' trasferito alla linea 130. La linea 140 stampa i valori ottenuti. Il 79 indica che e' stato riscontrato un "bad value".
Il -1 indica che l'errore si trova in un'istruzione. Il 9 e' la *gravita'-errore*. Il 110 indica che l'errore e' avvenuto alla linea 110.

```
>100 ON ERROR 130
>110 CALL SCREEN(18)
>120 STOP
>130 CALL ERR(W,X,Y,Z)
>140 PRINT W;X;Y;Z
>RUN

>  79 -1  9  110
```

---

# EXP

===

**Formato**

EXP(*espressione-numerica*)

**Descrizione**

La funzione EXP riporta il valore esponenziale ($e^x$) dell'*espressione-numerica*. Il valore di e e' 2.718281828459.

**Esempi**

Y=EXP(7) assegna ad Y il valore di e elevato alla settima potenza che e' 1096.633158429.

```
>100 Y=EXP(7)
```

L=EXP(4.394960467) assegna ad L il valore di e elevato alla potenza di 4.394960467 che e' 81.04142688868.

```
>100 L=EXP(4.394960467)
```

---

# FOR TO [STEP]

===

**Formato**

FOR *variabile-controllo* = *valore-iniziale* TO *limite*
    [STEP *incremento*]

**Descrizione**

L'istruzione FOR-TO-STEP ripete l'esecuzione delle istruzioni tra FOR-TO-STEP e NEXT finche' la *variabile-controllo* e' al di fuori della gamma compresa tra *valore-iniziale* e *limite*. L'istruzione FOR-TO-STEP e' utile per ripetere lo stesso incremento in un ciclo (loop). L'istruzione FOR-TO-STEP non puo' essere usata in un'istruzione IF-THEN-ELSE.
La *variabile-controllo* puo' essere qualsiasi variabile-numerica non indicizzata. Essa agisce come un contatore del ciclo. *Valore-iniziale* e *limite* sono espressioni numeriche. Il ciclo inizia con il *valore-iniziale* assegnato alla *variabile-controllo*. Nella seconda volta che viene eseguito il ciclo, il valore della *variabile-controllo* viene cambiato di uno o piu' a seconda di quanto specificato nell'*incremento*, il quale puo' essere un numero sia positivo che negativo. Questo continua finche' il valore della *variabile-controllo* e' al di fuori del valore del *limite*. Successivamente viene eseguita l'istruzione dopo il NEXT. Il valore della *variabile-controllo* non viene cambiato dopo la fine del ciclo.
Il valore della *variabile-controllo* puo' essere cambiato all'interno del ciclo, ma questa operazione deve essere compiuta con attenzione per evitare risultati imprevisti. I cicli possono essere "nidificati", cioe' possono trovarsi uno all'interno dell'altro. Si puo' uscire da un ciclo usando istruzioni del tipo di GOTO, GOSUB, IF-THEN-ELSE, per poi completare, eventualmente, l'esecuzione del ciclo. Tuttavia, non e' possibile iniziare un ciclo FOR-NEXT da punti diversi se non dal suo punto iniziale.
Se il *valore-iniziale* eccede il *limite* all'inizio del ciclo FOR-NEXT, nessuna delle istruzioni all'interno di esso vengono eseguite. L'esecuzione continua invece con la prima istruzione dopo il NEXT.

**Esempi**

FOR A=1 TO 5 STEP 2 esegue le istruzioni fra questo FOR e NEXT A tre volte, con A che assume i valori di 1, 3 e 5. Dopo che il ciclo e' terminato A ha un valore di 7.

```
>100 FOR A=1 TO 5 STEP 2
```

FOR J=7 TO -5 STEP -.5 esegue le istruzioni tra questo FOR e NEXT

```
>100 FOR J=7 TO -5 STEP -.5
```

J 25 volte, con J che assume i valori di 7, 6.5, 6, ...., -4, -4.5 e -5. Dopo che il ciclo e' finito J ha un valore di -5.5 .

**Programma**

Il programma a destra dimostra un uso dell'istruzione FOR-TO-STEP. Ci sono 3 cicli FOR-NEXT, con le *variabili-controllo* di CHAR, RIGA e COLONNA.

```
>100 CALL CLEAR
>110 D=0
>120 FOR CHAR=33 TO 63 STEP 3
 0
>130 FOR RIGA=1+D TO 21+D STE
 P 4
>140 FOR COLONNA=1+D TO 29+D
 STEP 4
>150 CALL VCHAR(RIGA,COLONNA,
 CHAR)
>160 NEXT COLONNA
>170 NEXT RIGA
>180 D=2
>190 NEXT CHAR
>200 GOTO 200
```

(Premere **FCTN 4** per fermare il programma).

---

# Sotttoprogramma GCHAR

---

**ormato**

CALL GCHAR(*riga,colonna,variabile-numerica*)

**escrizione**

:I sottoprogramma GCHAR legge un carattere in qualsiasi parte iello schermo. Il computer riporta nella *variabile-numerica* il :odice ASCII del carattere situato nella posizione specificata da *iga* e *colonna*.

*Riga* e *colonna* sono espressioni numeriche. Un valore di 1 per *iga* indica la parte superiore dello schermo. Un valore di 1 )er *colonna* indica il lato sinistro dello schermo. Lo schermo )uo' essere immaginato come la griglia disegnata qui di sotto.

**Esempi**

CALL GCHAR(12,16,X) assegna ad X il codice ASCII del carattere che si trova in riga 12 colonna 16.

```
>100 CALL GCHAR(12,16,X)
```

CALL GCHAR(R,C,K) mette in K il codice ASCII che si trova in riga R e colonna C.

```
>100 CALL GCHAR(R,C,K)
```

---

# GOSUB

---

**Formato**

GOSUB *numero-linea*
GO SUB *numero-linea*

**Descrizione**

L'istruzione GOSUB permette il trasferimento ad una subroutine. Quando e' eseguita, il controllo e' trasferito al *numero-linea* e quell' istruzione e le seguenti (le quali possono comprendere qualsiasi istruzione, comprese le istruzioni GOTO e altre GOSUB) vengono eseguite. Quando il programma incontra un'istruzione RETURN, il controllo viene riportato all'istruzione subito successiva all'istruzione GOSUB. Le subroutines sono molto utili quando e' richiesta la stessa serie di operazioni per piu' volte in un programma. Vedere anche ON...GOSUB. In TI Extended BASIC le subroutines possono richiamare se stesse. Gosub non puo' essere usata per trasferire il controllo da ed ad un sottoprogramma.

**Esempi**

GOSUB 200 trasferisce il controllo all'istruzione 200. Quest'istruzione e quelle fino a RETURN vengono eseguite, e poi il controllo ritorna alla istruzione successiva alla GOSUB.

```
>100 GOSUB 200
```

**Programma**

Il programma sulla destra illustra un uso di GOSUB. La subroutine della linea 260 calcola il fattoriale del valore di NUMB. L'intero programma calcola la soluzione dell'equazione:

$$NUMB = \frac{X!}{Y!*(X-Y)!}$$

dove il punto esclamativo sta per fattoriale. Questa formula e' usata per calcolare certe probabilita'. Per esempio, se si assegna ad X 52 e ad Y 5 , si trovera' il numero di cinque possibili mani a poker.

```
>100 CALL CLEAR
>110 INPUT "Enter X e Y:":X,Y
>120 IF X<Y THEN 110
>130 IF X>69 OR Y>69 THEN 110
>140 NUMB=X
>150 GOSUB 260
>160 NUMERATOR=NUMB
>170 NUMB=Y
>180 GOSUB 260
>190 DENOMINATOR=NUMB
>200 NUMB=X-Y
>210 GOSUB 260
>220 DENOMINATOR=DENOMINATOR*
 NUMB
>230 NUMB=NUMERATOR/DENOMINAT
 OR
>240 PRINT "IL NUMERO E'";NUM
 B
>250 STOP
>260 REM CALCOLO FATTORIALE
>270 IF NUMB<0 THEN PRINT "NE
 GATIVO" :: GOTO 110
>280 IF NUMB<2 THEN NUMB=1 ::
 GOTO 330
>290 MULT=NUMB-1
>300 NUMB=NUMB*MULT
>310 MULT=MULT-1
>320 IF MULT>1 THEN 300
>330 RETURN
```

---

# GOTO

---

**Formato**

GOTO *numero-linea*
GO TO *numero-linea*

**Descrizione**

L'istruzione GOTO vi permette di trasferire il controllo incondizionatamente ad un'altra linea all'interno del programma. Quando e' eseguita un'istruzione GOTO, il controllo e' trasferito alla prima istruzione della linea specificata dal *numero-linea*.

L'istruzione GOTO non dovrebbe essere usata per trasferire il controllo all'interno di sottoprogrammi.

Il programma sulla destra mostra l'uso di GOTO nella linea 160. Ogni volta che la linea viene raggiunta il programma esegue la linea 130 e dopo procede da questo nuovo punto.

```
>100 REM SOMMA 1 FINO A 100
>110 ANSWER=0
>120 NUMB=1
>130 ANSWER=ANSWER+NUMB
>140 NUMB=NUMB+1
>150 IF NUMB>100 THEN 170
>160 GOTO 130
>170 PRINT "LA RISPOSTA E' ";
 ANSWER
>RUN
LA RISPOSTA E' 5050
```

---

# Sottoprogramma HCHAR

---

**Formato**

CALL HCHAR(*riga,colonna,codice-carattere* [,*ripetizioni*])

**Descrizione**

Il sottoprogramma HCHAR visualizza un carattere in un punto qualsiasi dello schermo e opzionalmente lo ripete in orizzontale. Il carattere con il valore ASCII di *codice-carattere* viene collocato nella posizione specificata in *riga* e *colonna* e viene ripetuto orizzontalmente per tante volte quante ne sono specificate in *ripetizioni*.

Un valore di 1 per *riga* indica la parte superiore dello schermo. Un valore di 24 ne indica la parte inferiore. Un valore di 1 per *colonna* indica il lato sinistro dello schermo. Un valore di 32 ne indica il lato destro. Lo schermo puo' essere immaginato come la griglia raffigurata qui di sotto.

**Esempi**

CALL HCHAR(12,16,33) pone il carattere 33 (un punto !) in riga 12 e colonna 16.

```
>100 CALL HCHAR(12,16,33)
```

CALL HCHAR(1,1,ASC("!"),768) pone un punto ! a riga 1 e colonna 1, e lo ripete per 768 volte, riempiendo quindi tutto lo schermo.

```
>100 CALL HCHAR(1,1,ASC("!"),
 768)
```

CALL HCHAR(R,C,K,T) pone il carattere con il codice ASCII specificato dal valore di K in riga R, colonna C e lo ripete T volte.

```
>100 CALL HCHAR(R,C,K,T)
```

---

# IF THEN [ELSE]

===

**Formato**

IF *espressione-relazionale* THEN *num-linea1* [ELSE *num-linea2*]
IF *espressione-relazionale* THEN *istruz.N.1* [ELSE *istruz.N.2*]
IF *espressione-numerica* THEN *num-linea1* [ELSE *num-linea2*] IF *espressione-numerica* THEN *istruz.N.1* [ELSE *istruz.N.2*]

**Descrizione**
L'istruzione IF-THEN-ELSE vi permette di trasferire il controllo al *numero-linea 1* o eseguire l'*istruzione 1* se l'*espressione -relazionale* e' vera o se l'*espressione-numerica* non e' uguale a 0. Altrimenti il controllo passa all'istruzione seguente, o opzionalmente al *numero-linea 2* o all'*istruzione 2*.
*Istruzione 1* e *istruzione 2* possono essere ognuna composte da piu' istruzioni, separate tra loro dal simbolo separatore (::). Esse sono eseguite soltanto se e' eseguita la clausola che le precede immediatamente. L'istruzione IF-THEN-ELSE non puo' contenere DATA, DEF, DIM, FOR, NEXT, OPTION BASE, SUB o SUBEND.

**Esempi**

IF X>5 THEN GOSUB 300 ELSE X=X+5 opera come segue: se X e' maggiore di 5, viene eseguita la GOSUB 300. Quando la subroutine e' terminata, il controllo ritorna alla linea successiva a questa. Se X e' uguale o minore di 5, X e' posto uguale a X+5 ed il controllo passa alla linea successiva.

```
>100 IF X>5 THEN GOSUB 300 EL
 SE X=X+5
```

IF Q THEN C=C+1 :: GOTO 500 :: ELSE L=L/C :: GOTO 300 opera come segue: se Q non e' zero, allora C e' posto uguale a C+1 ed il controllo e' trasferito alla linea 500. Se Q e' zero, allora L e' posto uguale a L/C ed il controllo e' trasferito alla linea 300.

```
>100 IF Q THEN C=C+1::GOTO 50
 0::ELSE L=L/C::GOTO 300
```

IF A>3 THEN 300 ELSE A=0::GOTO 10 opera come segue: se A e' maggiore di 3, il controllo e' trasferito alla linea 300. Altrimenti A viene azzerato ed il controllo viene trasferito alla linea 10.

```
>100 IF A>3 THEN 300 ELSE A=0
 ::GOTO 10
```

IF A$="Y" THEN COUNT=COUNT+1::DISPLAY AT(24,1):"HERE WE GO AGAIN!":: GOTO 300 opera come segue: se A$ non e' uguale a "Y", il controllo passa alla linea successiva. Se A$ e' uguale a "Y", allora COUNT e' incrementato di 1, viene visualizzato un messaggio ed il controllo viene trasferito alla linea 300.

```
>100 IF A$="Y" THEN COUNT=COU
 NT+1::DISPLAY AT(24,1):"HERE
  WE GO AGAIN!"::GOTO 300
```

IF HOURS<=40 THEN PAY=HOURS*WAGE ELSE PAY=HOURS*WAGE+.5*WAGE*(HOURS -40)::OT=1 opera come segue: se HOURS e' minore o uguale a 40, a PAY viene assegnato il risultato di HOURS*WAGE ed il controllo passa alla linea successiva. Se HOURS e' maggiore di 40 allora a PAY viene assegnato il risultato di HOURS*WAGE+.5*WAGE*(HOURS-40), ad OT viene assegnato il valore 1, ed il controllo passa alla linea successiva.

```
>100 IF HOURS<=40 THEN PAY=HO
 URS*WAGE ELSE PAY=HOURS*WAGE
 +.5*WAGE*(HOURS-40)::OT=1
```

IF A=1 THEN IF B=2 THEN C=3 ELSE D=4 ELSE E=5 opera come segue: se A non e' uguale a 1, ad E viene assegnato il valore di 5 ed il controllo passa alla linea successiva. Se A e' uguale ad 1 e B e' diverso da 2, allora a D viene assegnato il valore 4 ed il controllo passa alla linea successiva. Se A e' uguale ad 1 e B e' uguale a 2, allora a C viene assegnato il valore 3 ed il controllo passa alla linea successiva.

```
>100 IF A=1 THEN IF B=2 THEN
 C=3 ELSE D=4 ELSE E=5
```

**Programma**

Il programma sulla destra illustra un uso di IF - THEN - ELSE. Esso accetta fino a 1000 numeri e poi li stampa in ordine crescente.

```
>100 CALL CLEAR
>110 DIM VALUE(1000)
>120 PRINT "INSERIRE IL VALOR
 E DA ORDINARE.":"DIGITARE
 '9999' PER FINIRE."
>130 FOR COUNT=1 TO 1000
>140 INPUT VALUE(COUNT)
>150 IF VALUE(COUNT)=9999 THE
 N 170
>160 NEXT COUNT
>170 COUNT=COUNT-1
>180 PRINT "ORDINAMENTO."
>190 FOR SORT1=1 TO COUNT-1
>200 FOR SORT2=SORT1+1 TO COU
 NT
>210 IF VALUE(SORT1)>VALUE(SO
 RT2)
>220 NEXT SORT2
>230 NEXT SORT1
>240 FOR SORTED=1 TO COUNT
>250 PRINT VALUE(SORTED)
>260 NEXT SORTED
```

# IMAGE

==================================================================

## Formato

IMAGE *formato-stringa*

## Descrizione

L'istruzione IMAGE specifica il formato nel quale i numeri devono esere stampati o visualizzati quando e' presente la clausola USING in PRINT o DISPLAY. Nessuna azione viene eseguita quando l'istruzione IMAGE e' incontrata durante l'esecuzione del programma. IMAGE deve essere l'unica istruzione su una linea. La descrizione successiva del *formato-stringa* e' valida anche nell'uso di un'esplicita IMAGE dopo la clausola USING nella PRINT...USING e nella DISPLAY...USING.
Il *formato-stringa* deve contenere 254 o meno caratteri e puo' essere composto da qualsiasi carattere. Essi sono trattati come segue:
Segni di sterlina (#) sono sostituiti dai valori della *lista di stampa* dati nelle PRINT...USING e DISPLAY...USING. Un segno di sterlina deve essere presente per ogni cifra del valore ed uno per il segno negativo se presente, oppure per ogni carattere che si vuole venga stampato. Se non c'e' abbastanza spazio per stampare il numero o i caratteri nello spazio stabilito, ogni segno di sterlina e' sostituito da un asterisco (*). Se dopo il punto decimale ci sono piu' numeri di quanti stabiliti dal numero di segno di sterlina (#) dopo il posto decimale nell'istruzione IMAGE, il numero viene arrotondato. Se ci sono meno caratteri non numerici di quelli stabiliti nella stringa di stampa, il valore stampato conterra' spazi vuoti al posto dei caratteri mancanti.
Per indicare che un numero deve essere espresso in notazione scientifica, devono essere dati accenti circonflessi (^) per E e per i numeri di potenza. Ci devono essere 4 o 5 circonflessi, e 10 o meno caratteri (segno di meno '-',segno di sterlina '#' e punto decimale '.') quando si usa il formato E.
Il punto decimale separa le parti intere e frazionali dei numeri, ed e' stampato dove esso appare nell'istruzione IMAGE.
Tutte le altre lettere, numeri e caratteri sono stampati esattamente come essi compaiono nell'istruzione IMAGE.
Il *formato-stringa* puo' essere racchiuso tra doppi apici. Se non lo e' vengono ignorati gli spazi iniziali e finali. Tuttavia, quando e' usato direttamente nella PRINT...USING e nella DISPLAY...USING deve essere racchiuso tra apici doppi.
Ogni istruzione IMAGE puo' avere spazio per molte "immagini", separate tra loro da qualsiasi carattere escluso il punto decimale. Se vengono dati piu' valori nell'istruzione PRINT...USING o DISPLAY...USING di quante siano le "immagini", allora queste vengono riusate a partire dall'inizio dell'istruzione.
Se si desidera si puo' mettere il *formato-stringa* direttamente nell'istruzione PRINT...USING o DISPLAY...USING immediatamente

preceduto da USING. Tuttavia, se si usa spesso il *formato-stringa* e' piu' efficiente riferirsi ad un'istruzione IMAGE.

**Esempi**

IMAGE $####.### permette di stampare qualsiasi numero da -999.999 a 9999.999 . Il seguente esempio mostra come alcuni semplici valori saranno stampati visualizzati.

```
>100 IMAGE $####.###
>110 PRINT USING 100:A
```

| *Valore* | *Stampa* |
|---|---|
| -999.999 | $-999.999 |
| -34.5 | $ -34.500 |
| 0 | $ 0.000 |
| 12.4565 | $ 12.457 |
| 6312.9991 | $6312.999 |
| 99999999 | $********* |

IMAGE LE RISPOSTE SONO ### E ##.## permette di stampare 2 numeri. Il primo puo' essere compreso tra -99 e 999 ed il secondo tra -9.99 e 99.99 . L'esempio che segue mostra come possono essere stampati o visualizzati alcuni semplici valori.

```
>200 IMAGE LE RISPOSTE SONO #
## E ##.##
>210 PRINT USING 200:A,B
```

| *Valori* | | *Stampa* |
|---|---|---|
| -99 | -9.99 | LE RISPOSTE SONO -99 E -9.99 |
| -7 | -3.459 | LE RISPOSTE SONO -7 E -3.46 |
| 0 | 0 | LE RISPOSTE SONO 0 E .00 |
| 14.8 | 12.75 | LE RISPOSTE SONO 15 E 12.75 |
| 795 | 852 | LE RISPOSTE SONO 795 E ***** |
| -984 | 64.7 | LE RISPOSTE SONO *** E 64.70 |

IMAGE CARO ####, permette di stampare una stringa di 4 caratteri. L'esempio che segue mostra come alcuni semplici valori possono essere stampati o visualizzati.

```
>300 IMAGE CARO ####,
>310 PRINT USING 300:X$
```

| *Valori* | *Stampa* |
|---|---|
| JOHN | CARO JOHN |
| TOM | CARO TOM |
| RALPH | CARO **** |

**Programmi**

Il programma sulla destra illustra un uso di IMAGE. Esso legge e stampa 7 numeri ed il loro totale. Le linee 110 e 120 preparano le immagini. Esse sono le stesse eccetto che per il segno del $ nella linea 110. Per mantenere lo spazio dove era il segno del $, il *formato-stringa* nella linea 120 e' racchiuso tra apici.

```
>100 CALL CLEAR
>110 IMAGE $####.##
>120 IMAGE " ####.##"
>130 DATA 233.45,-147.95,8.4,
 37.263,-51.299,85.2,464
>140 TOTAL=0
>150 FOR A=1 TO 7
>160 READ AMOUNT
>170 TOTAL=TOTAL+AMOUNT
```

La linea 180 stampa i valori usando le istruzioni IMAGE.

```
>180 IF A=1 THEN PRINT USING
 110:AMOUNT ELSE PRINT USING
 120:AMOUNT
>190 NEXT A
>200 PRINT " -------"
```

La linea 210 mostra che il formato puo' essere messo direttamente nell'istruzione PRINT...USING. I totali sono stampati con i punti decimali allineati.

```
>210 PRINT USING"$####.##":T
 OTAL
>RUN
$ 233.45
 -147.95
    8.40
   37.26
  -51.30
   85.20
  464.00
 -------
$ 629.06
```

Il programma sulla destra mostra l'effetto dell'uso di piu' valori nell'istruzione PRINT...USING di quante siano le immagini nella istruzione IMAGE.

```
>100 IMAGE ###.##,###.#
>110 PRINT USING 100:50.34,50
 .34,37.26,37.26
>RUN
 50.34, 50.3
 37.26, 37.3
```

---

# Sottoprogramma INIT

---

**Formato**

CALL INIT

**Descrizione**

Il sottoprogramma INIT e' usato insieme a LINK, LOAD e PEEK, per accedere ai sottoprogrammi in linguaggio Assembly. Il sottoprogramma INIT controlla che sia collegata l'espansione di memoria, prepara il computer ad eseguire programmi in linguaggio assembly, e carica una serie di routines di supporto nell'espansione di memoria.

Il sottoprogramma INIT deve essere chiamato prima dei sottoprogrammi LOAD e LINK. INIT rimuove qualsiasi sottoprogramma precedentemente caricato nell'espansione di memoria. Gli effetti di INIT durano finche' non viene spenta l'espansione di memoria e non ha bisogno di essere chiamato da ciascun programma che utilizza il sottoprogramma in questione.

Se l'espansione di memoria non e' collegata viene dato un errore di sintassi (SYNTAX ERROR).

# INPUT

==========================================================================

**Formato**

INPUT ["*informazione*":] *lista-variabili*

(Per informazioni sull'uso dell'istruzione INPUT con un *file* vedere INPUT con i *files*)

**Descrizione**

Questa forma dell'istruzione INPUT e' usata quando si inseriscono dati da tastiera. L'istruzione INPUT sospende l'esecuzione del programma finche' non sono inseriti i dati dalla tastiera. L'*informazione* facoltativa puo' visualizzare sullo schermo quale tipo di dati e' richiesto.
La *lista-variabili* contiene le variabili (normali o con indice, numeriche o di stringa) a cui devono essere assegnati i valori quando e' eseguita l'istruzione INPUT. Le variabili, se sono piu' di una, devono essere separate da virgole. Se un valore nella *lista-variabili* e' in ingresso, esso puo' essere usato successivamente come un indice nella stessa istruzione INPUT.
Quando si immettono valori di stringa, essi possono essere facoltativamente racchiusi tra apici. Tuttavia, se desiderate avere spazi iniziali, spazi finali o virgole l'intera stringa deve essere chiusa tra apici. Se deve essere immesso piu' di un valore essi devono essere separati da virgole.

**Opzioni**

L'*informazione* facoltativa deve essere seguita da un due punti ':'. E' visualizzata sullo schermo quando l'istruzione INPUT viene eseguita. Se non c'e' l'*informazione* vengono visualizzati un punto interrogativo ed uno spazio ad indicare che l'INPUT e' in attesa. Se c'e' l'*informazione*, essa prende il posto del punto interrogativo e dello spazio.

**Esempi**

| | |
|---|---|
| INPUT X permette l'ingresso di un numero. | >100 INPUT X |
| INPUT X$,Y permette l'ingresso di una stringa e di un numero. | >100 INPUT X$,Y |
| INPUT "INSERIRE 2 NUMERI:":A,B stampa l'*informazione* INSERIRE 2 NUMERI e permette l'immissione dei 2 numeri. | >100 INPUT "INSERIRE 2 NUMERI :":A,B |
| INPUT A(J),J prima valuta l'indice di A e poi accetta i dati in quel-l'indice di A. Poi viene accettato un valore in J. | >100 INPUT A(J),J |

INPUT J,A(J) prima accetta il dato in J e poi nell'elemento J della matrice A.

```
>100 INPUT J,A(J)
```

**Programma**

Il programma sulla destra illustra un uso di INPUT da tastiera. Le linee dalla 110 alla 140 permettono alla persona che usa il programma di inserire i dati come richiesto dalle informazioni.

Le linee dalla 170 alla 250 costruiscono una lettera basata sui dati immessi.

```
>100 CALL CLEAR
>110 INPUT "NOME     ?: ":NOM$
>120 INPUT "COGNOME ?: ":COG$
>130 INPUT "INSERIRE UN NUMER
 O DI 3 CIFRE: ":DOLLARS
>140 INPUT "INSERIRE UN NUMER
 O DI 2 CIFRE: ":CENT
>150 IMAGE OF $###.## E CIO'
 SE
>160 CALL CLEAR
>170 PRINT "CARO ";NOM$;"," :
 :
>180 PRINT "      QUESTA E' PE
 R RICORDARVI"
>190 PRINT "CHE SIETE IN DEBI
 TO DI"
>200 PRINT USING 150:DOLLARS+
 CENT/100
>210 PRINT "NON DOVETE PAGARC
 I, PRESTO "
>220 PRINT "RICEVERETE UNA LE
 TERA DAL NOSTRO"
>230 PRINT "PROCURATORE, INDI
 RIZZATA A "
>240 PRINT NOM$;" ";COG$;"!":
 :
>250 PRINT TAB(15);"CORDIALI
 SALUTI, ": : :TAB (15);"IL
 VOSTRO CREDITORE": : : :
>260 GOTO 260
```

(Premere **FCTN 4** per fermare il programma)

---

# INPUT (con i files)

---

**Formato**

INPUT #*numero-file* [,REC *numero-record*]: *lista-variabili*

(Per informazioni sull'uso dell'istruzione INPUT nell'inserimento dei dati da tastiera, vedere INPUT.)

**Descrizione**

L'istruzione INPUT, quando usata con i files, permette di leggere i dati dai files. L'istruzione INPUT puo' essere usata solo con files aperti in modo INPUT o UPDATE. I files in formato DISPLAY non possono avere piu' di 160 caratteri per ciascun record.

Il *numero-file* e la *lista-variabili* devono essere comprese nell'istruzione INPUT. Il *numero-record* puo' essere facoltativamente specificato quando si leggono files ad accesso casuale (RELATIVE) da dischetto.

Tutte le istruzioni che si riferiscono ai files si comportano cosi' con un *numero-file* compreso tra 0 e 255. Il *numero-file* e' assegnato ad un file particolare mediante l'istruzione OPEN. Il file numero 0 e' riservato alla tastiera ed allo schermo del computer. Esso non puo' essere usato per altri files ed e' sempre aperto. Il *numero-file* va introdotto con il segno di sterlina (#) seguito da un'*espressione-numerica* che viene arrotondata al numero intero piu' vicino, che e' un numero tra 0 e 255, ed e' il numero di un file aperto.

La *lista-variabili* e' la lista delle variabili nelle quali si desidera che siano posti i dati del file. Essa consiste in variabili numeriche o di stringa separate da virgole con una opzionale virgola in sospeso.

**Opzioni**

Si puo' facoltativamente specificare il numero del record da leggere con la specifica del *numero-record*. Esso puo' essere specificato solo per i files su disco che sono stati aperti in modo RELATIVE. Il primo record di un file e' il numero 0.

**Esempi**

INPUT #1:X$ mette nella variabile X$ il successivo valore disponibile nel file che era stato aperto con #1.

```
>100 INPUT #1:X$
```

INPUT #23:X,A,LL$ mette in X, A e LL$ i successivi 3 valori del file che era stato aperto con #23.

```
>100 INPUT #23:X,A,LL$
```

INPUT #11,REC 44:TAX mette in TAX il primo valore del record n. 44 del file che era stato aperto con #11.

```
>100 INPUT #11,REC 44:TAX
```

INPUT #3:A,B,C, mette in A ,B e C i successivi 3 valori del file che era stato aperto con #3. La virgola dopo C crea una condizione di ingresso in sospeso. Quando viene eseguita la successiva istruzione INPUT o LINPUT che usa questo file, avviene una delle seguenti azioni: se la successiva istruzione INPUT o LINPUT non ha la clausola REC, il computer usa i dati a cominciare dal punto in cui si era fermata la precedente istruzione INPUT. Se l'istruzione successiva INPUT o LINPUT comprende la clausola REC, il computer termina la condizione di ingresso in sospeso e legge il record specificato.

```
>100 INPUT #3:A,B,C,
```

**Programma**

Il programma sulla destra illustra un uso dell'istruzione INPUT. Esso apre un file sul registratore a cassette e scrive 5 records nel file. Poi torna indietro, legge i records e li visualizza sullo schermo.

```
>100 OPEN #1:"CS1",SEQUENTIAL
 ,INTERNAL,OUTPUT,FIXED 64
>110 FOR A=1 TO 5
>120 PRINT #1:"QUESTO E' IL R
 ECORD",A
>130 NEXT A
>140 CLOSE #1
>150 CALL CLEAR
>160 OPEN #1:"CS1",SEQUENTIAL
 ,INTERNAL,INPUT,FIXED 64
>170 FOR B=1 TO 5
>180 INPUT #1:A$,C
>190 DISPLAY AT(B,1):A$;C
>200 NEXT B
>210 CLOSE #1
>RUN
 * REWIND CASSETTE TAPE   CS1
   THEN PRESS ENTER
 * PRESS CASSETTE RECORD  CS1
   THEN PRESS ENTER
 * PRESS CASSETTE STOP    CS1
   THEN PRESS ENTER
 * REWIND CASSETTE TAPE   CS1
   THEN PRESS ENTER
 * PRESS CASSETTE PLAY    CS1
   THEN PRESS ENTER
 QUESTO E' IL RECORD     1
 QUESTO E' IL RECORD     2
 QUESTO E' IL RECORD     3
 QUESTO E' IL RECORD     4
 QUESTO E' IL RECORD     5
 * PRESS CASSETTE STOP    CS1
   THEN PRESS ENTER
```

Vedere il manuale del sistema di memoria a dischi per le istruzioni con l'uso dei dischetti.

---

# INT

---

**Formato**

INT (*espressione-numerica*)

**Descrizione**

La funzione INT restituisce l'intero piu' grande, piu' piccolo o uguale all'*espressione-numerica*.

**Esempi**

| | |
|---|---|
| PRINT INT(3.4) stampa 3. | >100 PRINT INT(3.4) |
| X=INT(3.9) pone X uguale a 3. | >100 X=INT(3.90) |
| P=INT(3.9999999999) pone P uguale a 3. | >100 P=INT(3.9999999999) |
| DISPLAY AT(3,7):INT(4.0) visualizza 4 a riga 3, colonna 7. | >100 DISPLAY AT(3,7):INT(4.0) |
| N=INT(-3.9) pone N uguale a -4. | >100 N=INT(-3.9) |
| K=INT(-3.0000001) pone K uguale a -4. | >100 K=INT(-3.0000001) |

# Sottoprogramma JOYST

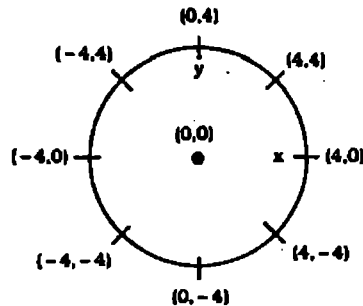
==================================================================

**Formato**

CALL JOYST (*unita'-tastiera, X-ritorno, Y-ritorno* )

**Descrizione**

Il sottoprogramma JOYST riporta in *X-ritorno* e in *Y-ritorno* i dati basati sulla posizioni del joystick nel comando a distanza (venduto separatamente), riferite all'*unita'-tastiera*. L'*unita'-tastiera* e' un'espressione numerica con valore da 1 a 4. I valori 1 e 2 sono i joystick 1 e 2. I valori 3 e 4 sono riservati a modi particolari per la tastiera della consolle.

I valori restituiti in *X-ritorno* e *Y-ritorno* dipendono dalla posizione del joystick. I valori restituiti sono raffigurati qui di sotto. Il primo valore tra parentesi e' posto in *X-ritorno* ed il secondo in *Y-ritorno*.

**Esempi**

CALL JOYST(1,X,Y) riporta dei valori in X e Y in relazione alla posizione del joystick numero 1.

```
>100 CALL JOYST(1,X,Y)
```

**Programma**

Il programma sulla destra illustra un uso del sottoprogramma JOYST. Esso crea uno sprite e lo muove in funzione del movimento del JOYSTICK.

```
>100 CALL CLEAR
>110 CALL SPRITE(#1,33,5,96,1
 28)
>120 CALL JOYST(1,X,Y)
>130 CALL MOTION(#1,-Y,X)
>140 GOTO 120
```

(Premere **FCTN 4** per fermare il programma).

---

# Sottoprogramma KEY

---

**Formato**

CALL KEY (*unita'-tastiera, variabile-ritorno, variabile-stato*)

**Descrizione**

Il sottoprogramma KEY assegna il codice del tasto premuto alla *variabile-ritorno*. Il valore assegnato dipende dall'*unita'-tastiera* specificata. Se l'*unita'-tastiera* e' 0 viene controllata l'intera tastiera, ed il valore posto nella *variabile-ritorno* e' il codice ASCII del tasto premuto. Se non e' premuto nessun tasto la *variabile-ritorno* viene posta uguale a -1. Vedere l'Appendice C per una lista dei codici ASCII.
Se l'*unita'-tastiera* e' 1, viene controllata la parte sinistra della tastiera. Se l'*unita'-tastiera* e' 2, viene controllata la parte destra della tastiera. I possibili valori posti nella *variabile-ritorno* si trovano nell'Appendice J. I valori 3, 4 e 5 sono riservati a particolari usi della tastiera. Con l'unita' 3 i caratteri alfabetici sono soltanto maiuscoli ed i tasti di funzione producono codici da 1 a 15. L'unita' 4 ridefinisce la tastiera in modo Pascal e i tasti di funzione producono codici da 129 a 143. L'unita' 5 ridefinisce la tastiera in modo BASIC e i codici dei tasti funzione vanno da 1 a 15. I tasti di controllo non sono attivati con l'unita' 3, vanno da 1 a 31 con l'unita' 4 e da 128 a 159 (e 187) con l'unita' 5. Riferirsi al manuale d'uso fornito con la tastiera per ulteriori chiarimenti.
La *variabile-ritorno* deve essere una variabile numerica, e contiene il codice numerico del tasto premuto.
La *variabile-stato* indica che un tasto e' stato premuto. Un valore di 1 significa che un nuovo tasto e' stato premuto dopo l'ultima CALL KEY eseguita. Un valore di -1 significa che lo stesso tasto era stato premuto nella precedente CALL KEY. Un valore di 0 indica che non e' stato premuto alcun tasto.

**Esempio**

CALL KEY(0,K,S) riporta in K il codice ASCII di ciascun tasto premuto, ed in S un valore che indica se e' stato premuto o no un tasto.

```
>100 CALL KEY(0,K,S)
```

**Programma**

Il programma sulla destra illustra un uso del sottoprogramma KEY. Esso crea uno sprite e poi lo muove in relazione ai tasti premuti sulla parte sinistra della tastiera.
Notare che la linea 130 riporta il controllo alla linea 120 se non e' stato premuto alcun tasto.

```
>100 CALL CLEAR
>110 CALL SPRITE(#1,33,5,96,1
 28)
>120 CALL KEY(1,K,S)
>130 IF S=0 THEN 120
>140 IF K=5 THEN Y=-4
>150 IF K=0 THEN Y=4
>160 IF K=2 THEN X=-4
>170 IF K=3 THEN X=4
>180 IF K=1 THEN X,Y=0
>190 IF K>5 THEN X,Y=0
>200 CALL MOTION(#1,Y,X)
>210 GOTO 120
```

(Premere **FCTN 4** per fermare il programma).

---

# LEN

---

**Formato**

LEN(*espressione-stringa*)

**Descrizione**

La funzione LEN restituisce il numero dei caratteri della *espressione-stringa*. Uno spazio viene contato come un carattere.

**Esempi**

PRINT LEN("ABCDE") stampa 5.

```
>100 PRINT LEN("ABCDE")
```

X=LEN("QUESTA E' UNA FRASE") assegna ad X il valore 19.

```
>100 X=LEN("QUESTA E' UNA FRA
 SE")
```

DISPLAY LEN("") visualizza 0.

```
>100 DISPLAY LEN("")
```

DISPLAY LEN(" ") visualizza 1.

```
>100 DISPLAY LEN(" ")
```

---

# LET

---

**Formato**

[LET] *variabile-numerica* [,*variab-numerica*,..]=*espr-numerica*
[LET] *variabile-stringa* [,*variab-stringa*,.. ]=*espr-stringa*

**Descrizione**

L'istruzione LET assegna il valore di un'espressione alla variabile(i) specificata(e). Il computer valuta l'espressione sulla destra e mette il suo valore nelle variabili a sinistra. Se sono specificate piu' variabili sulla sinistra esse vanno separate da virgole. LET e' facoltativo, ed e' omesso negli esempi di questo manuale. Tutti gli indici delle variabili sulla sinistra sono valutati prima che sia eseguita qualsiasi assegnazione.

Potete usare operatori relazionali e logici nell'*espressione-numerica*. Se il valore logico o relazionale e' vero, viene assegnato un valore di -1 alla *variabile-numerica*. Se falsa, viene assegnato 0.

**Esempi**

| | |
|---|---|
| T=4 pone il valore 4 in T. | >100 T=4 |
| X,Y,Z=12.4 pone il valore 12.4 in X, Y e Z. | >100 X,Y,Z=12.4 |
| A=3<5 pone -1 in A poiche' e' vero che 3 e' minore di 5. | >100 A=3<5 |
| B=12<7 pone 0 in B dato che non e' vero che 12 e' minore di 7. | >100 B=12<7 |
| I,A(I)=3 pone 3 in A(I) con un qualsiasi valore che era stato posto in I, e poi pone 3 in I. | >100 I,A(I)=3 |
| L$,D$,B$="B" pone "B" in L$, D$ e B$. | >100 L$,D$,B$="B" |

---

# Sottoprogramma LINK

---

**Formato**

CALL LINK(*nome-sottoprogramma* [,*argomenti*])

**Descrizione**

Il sottoprogramma LINK e' usato insieme a INIT, LOAD e PEEK per accedere a sottoprogrammi in linguaggio Assembly. Il sottoprogramma LINK passa il controllo e, facoltativamente, una lista di parametri da un programma TI Extended BASIC ad un sottoprogramma in linguaggio Assembly.

Il *nome-sottoprogramma* e' il nome del sottoprogramma che deve essere chiamato. Esso deve essere stato precedentemente caricato nell'espansione di memoria con un comando o istruzione CALL LOAD. Gli *argomenti* sono una lista di variabili ed espressioni che si riferiscono a quanto richiesto dallo specifico sottoprogramma in linguaggio Assembly che e' stato chiamato.

---

# LINPUT

---

**Formato**

LINPUT *#numero-file* [,REC *numero-record*] :*variabile-stringa*
LINPUT [*informazione*:] *variabile-stringa*

**Descrizione**

L'istruzione LINPUT permette l'assegnazione di un'intera linea, di un record o (se c'e' un record con ingresso in sospeso) la restante parte di un record alla *variabile-stringa*. Non viene eseguito alcun controllo su cio' che viene immesso, cosicche' virgole, spazi iniziali e finali, punti e virgole, punti e apici sono posti nella *variabile-stringa* cosi' come si trovano.

**Opzioni**

Un *#numero-file* puo' essere specificato. Se il file e' in formato RELATIVE, un record specifico puo' essere specificato con REC. Il file deve essere un file di tipo DISPLAY. Se non e' specificato alcun file, una *informazione* puo' essere visualizzata prima dell'accettazione dei dati in input da tastiera.

**Esempi**

LINPUT L$ assegna a L$ qualsiasi cosa digitata prima di premere ENTER.

```
>100 LINPUT L$
```

LINPUT "NOME: ":NM$ visualizza NOME: e assegna ad NM$ qualsiasi cosa digitata prima di premere ENTER.

```
>100 LINPUT "NOME: ":NM$
```

LINPUT #1,REC M:L$(M) assegna ad L$(M) il valore che si trovava nel record M del file che era stato aperto con #1.

```
>100 LINPUT #1,REC M:L$(M)
```

**Programma**

Il programma sulla destra illustra l'uso di LINPUT. Esso legge un file gia' esistente e visualizza solo le linee che contengono la parola "GLI".

```
>100 OPEN #1:"DSK1.TEXT1",INP
 UT,FIXED 80,DISPLAY
>110 IF EOF(1) THEN CLOSE #1
 :: STOP
>120 LINPUT #1:A$
>130 I=POS(A$,"GLI",1)
>140 IF I<>0 THEN PRINT A$
>150 GOTO 110
```

---

# LIST

---

**Formato**

LIST ["*nome-periferica*":][*numero-linea*]
LIST ["*nome-periferica*":][*numero-linea iniz*]-[*num-linea fin*]

**Descrizione**

Il comando LIST vi permette di visualizzare le lineè di programma. Se LIST non e' seguito da nessun numero sara' listato l'intero programma presente in memoria. Se seguito da un numero, sara' listata la linea con quel numero. Se un numero e' seguito dal segno meno viene listata quella linea e tutte quelle seguenti. Se un numero e' preceduto dal segno meno sono listate tutte le linee del programma dall'inizio fino alla linea specificata. Se dopo LIST sono specificati due numeri separati tra loro dal segno meno vengono listate sullo schermo le linee comprese fra i due numeri.

Premendo un tasto qualsiasi mentre scorre il listato sullo schermo e' possibile arrestarne lo scorrimento per poi riprenderlo premendo nuovamente lo stesso o un altro tasto. Allo stesso modo si blocca DEFINITIVAMENTE il listing premendo **FCTN 4** (CLEAR).

**Opzioni**
Normalmente il listato appare sullo schermo. Se desiderate invece dirigerlo su qualche periferica come ad esempio la stampante o l'interfaccia RS232 specificare il *nome-periferica* dopo il comando LIST.
**Esempi**

| | |
|---|---|
| LIST lista sullo schermo l'intero programma in memoria. | >LIST |
| LIST 100 lista la linea 100. | >LIST 100 |
| LIST 100- lista la linea 100 e tutte le successive. | >LIST 100- |
| LIST -200 lista tutte le linee precedenti fino alla 200 compresa. | >LIST -200 |
| LIST 100-200 lista tutte le linee comprese fra la 100 e la 200. | >LIST 100-200 |
| LIST "PIO" lista l'intero programma sulla stampante specificata. | >LIST "PIO" |
| LIST "PIO":-200 lista tutte le linee dall'inizio fino alla 200 inclusa, sulla stampante. | >LIST "PIO":-200 |

# Sottoprogramma LOAD

**Formato**

CALL LOAD("*nome-accesso*"[,*indirizzo*,*byte1*[,...],
*campo-file*,...])

**Descrizione**

Il sottoprogramma LOAD e' usato, insieme ad INIT, LINK e PEEK, per accedere a sottoprogrammi in linguaggio Assembly. Il sottoprogramma LOAD carica nell'espansione di memoria o un file Assembly in codice oggetto o direttamente dei dati da utilizzare successivamente con l'istruzione LINK.

Il sottoprogramma LOAD puo' specificare uno o piu' files da cui caricare i dati oggetto o l'elenco dei dati da caricare direttamente, che consistono in un *indirizzo* seguito dai *bytes* dei dati. L'*indirizzo* ed i *bytes* riferiti ai dati vanno separati da virgole. I dati che vanno inseriti direttamente in memoria vanno separati dal *campo-file*, che e' un'espressione di stringa che specifica un file dal quale caricare il codice oggetto in linguaggio Assembly. Il *campo-file* puo' essere una stringa nulla quando viene usato semplicemente per separare i campi dei dati da inserire direttamente. L'uso del sottoprogramma LOAD con valori errati puo' causare un malfunzionamento del computer o un suo arresto che rende necessario lo spegnimento di esso per alcuni istanti prima di riaccenderlo.

I nomi dei sottoprogrammi in linguaggio Assembly sono inclusi nel file (vedere LINK).

# Sottoprogramma LOCATE

**Formato**

CALL LOCATE(*numero-sprite,pixel-riga,pixel-colonna* [,...])

**Descrizione**

Il sottoprogramma LOCATE e' usato per cambiare la posizione degli sprite data in *pixel-riga* e *pixel-colonna*. *Pixel-riga* e *pixel-colonna* sono numerati progressivamente a partire dall'angolo superiore sinistro dello schermo. *Pixel-riga* va da 1 a 192 e *pixel-colonna* va da 1 a 256. In realta' i *pixel-riga* possono arrivare fino a 256 ma le locazioni da 193 a 256 sono al di sotto del margine inferiore dello schermo. La posizione dello sprite e' l'angolo superiore sinistro del carattere(i) che lo definiscono.

**Programma**

Il programma sulla destra illustra l'uso del sottoprogramma LOCATE.
La linea 110 crea uno sprite con un punto esclamativo rosso che si muove con una certa velocita'.
La linea 140 pone lo sprite in una posizione scelta casualmente nelle linee 120 e 130.
La linea 150 ripete l'esecuzione.

```
>100 CALL CLEAR
>110 CALL SPRITE(#1,33,7,1,1,
 25,25)
>120 YLOC=INT(RND*192+1)
>130 XLOC=INT(RND*256+1)
>140 CALL LOCATE(#1,YLOC,XLOC
 )
>150 GOTO 120
```

(Premere **FCTN 4** per fermare il programma).

Vedere il terzo esempio del sottoprogramma SPRITE.

---

# LOG

---

**Formato**

LOG(*espressione-numerica*)

**Descrizione**

La funzione LOG restituisce il logaritmo naturale dell'*espressione-numerica* dove questa e' maggiore di zero (0).
La funzione LOG e' l'inverso della funzione EXP.

**Esempi**

PRINT LOG(3.4) stampa il logaritmo naturale di 3.4 che e' 1.223775431622 .

```
>100 PRINT LOG(3.4)
```

X=LOG(EXP(7.2)) pone X uguale al logaritmo naturale di *e* elevato a 7.2 che e' 7.2 .

```
>100 X=LOG(EXP(7.2))
```

S=LOG(SQR(T)) pone S uguale al logaritmo naturale della radice quadrata del valore di T.

```
>100 S=LOG(SQR(T))
```

**Programma**

Il programma a destra restituisce il logaritmo naturale di ogni numero positivo con una qualsiasi base.

```
>100 CALL CLEAR
>110 INPUT "BASE: ":B
>120 IF B<1 THEN 110
>130 INPUT "NUMERO: ":N
>140 IF N<0 THEN 130
>150 LG=LOG(N)/LOG(B)
>160 PRINT "LOG BASE";B;"DI";
 N;"E'";LG
>170 GOTO 110
```

(Premere **FCTN 4** per fermare il programma).

---

# Sottoprogramma MAGNIFY

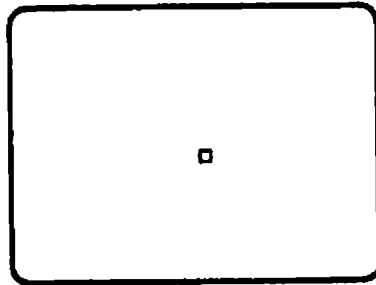
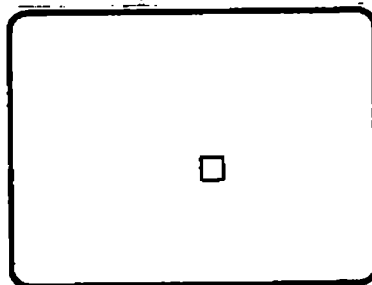
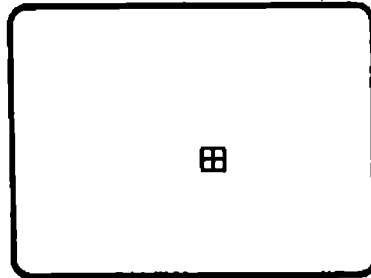
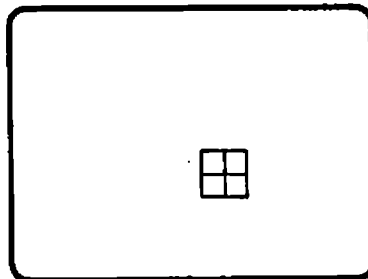
---

**Formato**

CALL MAGNIFY(*fattore-ingrandimento*)

**Descrizione**

Il sottoprogramma MAGNIFY vi permette di specificare la dimensione degli sprites e di quanti caratteri e' costituito ciascuno sprite. MAGNIFY ha effetto su tutti gli sprites. Il *fattore-ingrandimento* puo' essere 1, 2, 3 o 4. Se in un programma non compare alcun CALL MAGNIFY, il valore di default e' 1.

Un *fattore-ingrandimento* di 1 fa si che tutti gli sprites abbiano un'unica dimensione e non vengano ingranditi. Questo significa che ciascuno sprite e' definito solo dal carattere specificato quando e' stato creato e accetta solo una posizione del carattere sullo schermo.

Un *fattore-ingrandimento* di 2 fa si che tutti gli sprites abbiano un'unica dimensione e siano ingranditi. Questo significa che ciascuno sprite e' definito solo dal carattere specificato quando e' stato creato, ma occupa fino a 4 posizioni sullo schermo. Ciascun carattere definito viene ingrandito per quattro volte sullo schermo. L'espansione da un *fattore-ingrandimento* di 1 si trova in basso a destra.

Un *fattore-ingrandimento* di 3 fa in modo che tutti gli sprites abbiano una misura doppia e non ingranditi. Questo significa che ogni sprite e' definito da 4 posizioni di carattere che comprendono il carattere specificato. Il primo carattere e' quello specificato quando lo sprite e' stato creato se il suo numero e' divisibile per 4 o il numero precedente divisibile per 4. Quel carattere e' quello nella parte superiore sinistra dello sprite. Il carattere seguente e' quello nella parte inferiore sinistra. Il successivo e' nella parte superiore destra dello sprite. L'ultimo carattere e' nella parte inferiore destra. Il carattere specificato quando lo sprite e' stato creato e' uno dei quattro che compongono lo sprite. Lo sprite occupa 4 locazioni dello schermo.

Un *fattore-ingrandimento* di 4 fa in modo che tutti gli sprites siano di misura doppia ed ingranditi. Questo significa che ciascuno sprite e' definito da 4 posizioni sullo schermo che comprendono il carattere specificato. Il primo carattere e' quello specificato quando e' stato creato lo sprite se il suo numero e' divisibile per 4 o il precedente numero divisibile per 4. Quel carattere e' nella parte superiore sinistra dello sprite. Il successivo e' nella parte inferiore sinistra. Il carattere seguente e' nella parte superiore destra dello sprite. L'ultimo e' nella parte superiore destra. Il carattere specificato quando e' stato creato lo sprite e' uno dei quattro che compongono lo sprite. Lo sprite occupa 16 locazioni dello schermo. L'espansione da un *fattore-ingrandimento* di 3 e' in basso a destra.

**Programma**

Il seguente programma illustra un uso del sottoprogramma MAGNIFY. Quando esso viene eseguito appare una piccola figura vicino al centro dello schermo. In un momento esso diventa due volte piu' grande occupando 4 locazioni dello schermo. In un altro momento esso viene sostituito dall'angolo superiore sinistro di una figura piu' grande, che occupa ancora 4 locazioni dello schermo. Poi appare la figura completa, che occupa 16 locazioni dello schermo. Infine essa si riduce nelle dimensioni di 4 locazioni.

```
>100 CALL CLEAR
>110 CALL CHAR(96,"1898FF3D3C
 3CE404")
>120 CALL SPRITE(#1,96,5,92,1
 24)
>130 GOSUB 230
>140 CALL MAGNIFY(2)
>150 GOSUB 230
>160 CALL CHAR(96,"0103C3417F
 3F07070707077E7C40000080C0C0
 80FCFEE2E3E0E0E06060606070")
>170 GOSUB 230
>180 CALL MAGNIFY(4)
>190 GOSUB 230
>200 CALL MAGNIFY(3)
>210 GOSUB 230
>220 STOP
>230 REM PAUSA
>240 FOR PAUSA=1 TO 500
>250 NEXT PAUSA
>260 RETURN
```

La linea 110 definisce il carattere 96.
La linea 120 accende uno sprite usando il carattere 96. Per default il fattore d'ingrandimento e' 1.
La linea 140 porta a 2 il fattore d'ingrandimento.
La linea 160 ridefinisce il carattere 96. Poiche' la definizione e' lunga 64 caratteri, essa definisce anche i caratteri 97, 98 e 99.
La linea 180 porta il fattore d'ingrandimento a 4.
La linea 200 porta il fattore d'ingrandimento a 3.

---

# MAX

---

**Formato**

MAX (*espressione-numerica1,espressione-numerica2*)

**Descrizione**

La funzione MAX restituisce il valore piu' grande tra *espressione-numerica1* e *espressione-numerica2*. Se esse sono uguali viene restituito il loro valore.

**Esempi**

| | |
|---|---|
| PRINT MAX(3,8) stampa 8. | >100 PRINT MAX(3,8) |
| F=MAX(3E12,1800000) assegna ad F il valore 3E12. | >100 F=MAX(3E12,1800000) |
| G=MAX(-12,-4) pone G uguale a -4. | >100 G=MAX(-12,-4) |
| L=MAX(A,B) pone L uguale a 7 se A e' 7 e B e', ad esempio -5. | >100 L=MAX(A,B) |

---

# MERGE

---

**Formato**

MERGE ["]*periferica.nome-file*["]

**Descrizione**

Il comando MERGE fonde le linee del *nome-file* dalla *periferica* specificata, con le linee del programma gia' presente nella memoria del computer. Se un numero di linea del *nome-file* e' uguale al *numero-linea* del programma in memoria la nuova linea rimpiazza la vecchia. Altrimenti le righe sono inserite in ordine di numero tra le linee gia' presenti in memoria. Il comando MERGE non rimuove i breakpoints. Inoltre MERGE puo' essere usato soltanto con i dischetti.

Nota: i files possono essere fusi in memoria se sono stati salvati con l'opzione MERGE. Vedere il comando SAVE per maggiori informazioni.

**Esempio**

MERGE DSK1.SUB fonde il programma SUB con il programma attualmente in memoria.

```
>MERGE DSK1.SUB
```

**Programma**

Se il programma a destra viene salvato sul DSK1 come BOUNCE con l'opzione MERGE, esso puo' essere fuso con altri programmi come mostrato nella pagina successiva.

```
>100 CALL CLEAR
>110 RANDOMIZE
>140 DEF RND50=INT(RND-25)
>150 GOSUB 10000
>10000 FOR AA=1 TO 20
>10010 QQ=RND50
>10020 LL=RND50
>10030 CALL MOTION(#1,QQ,LL)
>10040 NEXT AA
>10050 RETURN


>SAVE "DSK1.BOUNCE",MERGE
```

Il programma sulla destra puo' essere inserito nella memoria del computer.

```
>120 CALL CHAR(96,"18183CFFFF
 3C1818")
>130 CALL SPRITE(#1,96,7,92,1
 28)
>150 GOSUB 500
>160 STOP
```

Ora potete fondere il programma BOUNCE con il programma di sopra.

```
>MERGE DSK1.BOUNCE
```

Il programma che risulta dalla fusione di BOUNCE con il programma di sopra e' mostrato qui a destra.

Notare che la linea 150 viene dal programma che era stato fuso, e non dal programma che era in memoria.

```
>LIST
>100 CALL CLEAR
>110 RANDOMIZE
>120 CALL CHAR(96,"18183CFFFF
 3C1818")
>130 CALL SPRITE(#1,96,7,92,1
 28)
>140 DEF RND50=INT(RND-25)
>150 GOSUB 10000
>160 STOP
>10000 FOR AA=1 TO 20
>10010 QQ=RND50
>10020 LL=RND50
>10030 CALL MOTION(#1,QQ,LL)
>10040 NEXT AA
>10050 RETURN
```

# MIN

**Formato**

MIN (*espressione-numerica1,espressione-numerica2*)

**Descrizione**

La funzione MIN restituisce il piu' piccolo dei valori tra *espressione-numerica1* e *espressione-numerica2*. Se essi sono uguali e' restituito il loro valore.

**Esempi**

| | |
|---|---|
| PRINT MIN(3,8) stampa 3. | >100 PRINT MIN(3,8) |
| F=MIN(3E12,1800000) pone F uguale a 1800000. | >100 F=MIN(3E12,1800000) |
| G=MIN(-12,-4) pone G uguale a -12. | >100 G=MIN(-12,-4) |
| L=MIN(A,B) pone L uguale a -5 se ad esempio, A e' uguale a 7 e B e' uguale a -5 . | >100 L=MIN(A,B) |

---

# Sottoprogramma MOTION

---

**Formato**

CALL MOTION(#*num. sprite*,*velocita'-riga*,*velocita'-colonna*[,...])

**Descrizione**

Il sottoprogramma MOTION e' usato per specificare la *velocita' di riga* e la *velocita' di colonna* di uno sprite. Se entrambe le velocita' sono 0 lo sprite e' fermo. Un numero positivo di *velocita' di riga* sposta lo sprite in basso, ed un valore negativo lo muove verso l'alto. Un numero positivo di *velocita' di colonna* muove lo sprite verso destra ed un valore negativo lo muove verso sinistra. Se entrambe le velocita' sono diverse da 0 lo sprite si muove lentamente verso un angolo in una direzione determinata dai valori di quel momento.

Le *velocita' di riga e di colonna* possono variare da - 128 a 127. Un valore vicino a 0 e' molto lento. Un valore molto piu' grande di 0 e' molto veloce. Quando uno sprite arriva all'estremita' dello schermo esso sparisce e riappare nella posizione corrispondente sull'altro lato dello schermo.

**Programma**

| | |
|---|---|
| Il programma a destra mostra un uso del sottoprogramma MOTION. | >100 CALL CLEAR |
| La linea 110 crea uno sprite. | >110 CALL SPRITE(#1,33,5,92,1<br>24) |
| Le linee 120 e 130 assegnano i valori al movimento dello sprite. | >120 FOR XVEL=-16 TO 16 STEP<br>2<br>>130 FOR YVEL=-16 TO 16 STEP<br>2 |
| La linea 140 visualizza i valori attuali del movimento dello sprite. | >140 DISPLAY AT(12,11):XVEL;Y<br>VEL |
| La linea 150 pone lo sprite in movimento. | >150 CALL MOTION(#1,YVEL,XVEL<br>) |
| Le linee 160 e 170 completano il ciclo che regola i valori per il movimento dello sprite. | >160 NEXT YVEL<br>>170 NEXT XVEL |

---

## NEW

---

**Formato**

NEW

**Descrizione**

Il comando NEW pulisce sia la memoria che lo schermo e prepara il computer per un nuovo programma. Tutti i valori sono azzerati e tutti i caratteri definiti ridiventano non definiti. Tutti i files vengono chiusi. I caratteri da 32 a 45 sono riportati alla loro rappresentazione standard. I comandi TRACE e BREAK sono annullati.

Assicurarsi di aver salvato il programma, sul quale state lavorando, prima di immettere NEW poiche' esso non e' piu' recuperabile una volta che NEW e' stato eseguito.

# NEXT

**Formato**

NEXT *variabile-controllo*

Vedere ON BREAK, ON WARNING e RETURN (con ON ERROR) per l'uso di NEXT con queste istruzioni.

**Descrizione**

L'istruzione NEXT e' sempre accompagnata dall'istruzione FOR-TO-STEP nella costruzione di un ciclo. La *variabile-controllo* deve essere la stessa *variabile-controllo* dell'istruzione FOR-TO-STEP. L'istruzione NEXT non puo' comparire in un'istruzione IF-THEN-ELSE.

L'istruzione NEXT controlla che il ciclo sia ripetuto. Ogni volta che viene eseguita l'istruzione NEXT, la *variabile-controllo* e' cambiata in base al valore specificato dopo STEP nell'istruzione FOR-TO-STEP, o di 1 se non c'e' la clausola STEP. Se il valore della *variabile-controllo* e' compreso tra valore iniziale e limite, il ciclo e' eseguito di nuovo. Altrimenti il controllo passa all'istruzione successiva al NEXT. Cosi', il valore della *variabile-controllo* alla fine del ciclo e' sempre il primo valore ottenuto oltre i valori stabiliti. Vedere l'istruzione FOR-TO-STEP per maggiori informazioni.

**Programma**

Il programma a destra illustra un uso dell'istruzione NEXT nelle linee 130 e 140.

```
>100 TOTAL=0
>110 FOR COUNT=10 TO 0 STEP -
 2
>120 TOTAL=TOTAL+COUNT
>130 NEXT COUNT
>140 FOR PAUSA=1 TO 100::NEXT
 PAUSA
>150 PRINT TOTAL,COUTN;PAUSA
>RUN
  30            -2  101
```

---

# NUMBER

---

**Formato**

NUMBER [*linea-iniziale*] [,*incremento*]
NUM [*linea-iniziale*] [,*incremento*]

**Descrizione**

Il comando NUMBER produce numeri di linea in sequenza, permettendo l'entrata di linee di programma senza dover digitare i numeri di linea. Se la *linea-iniziale* e l'*incremento* non sono specificati, i numeri di linea partono da 100 con l'incremento di 10. Potete dare il comando in ciascun momento in modo Comando. Se una linea esiste gia' essa viene visualizzata. Potete scrivere al di sopra di essa, sostituirla o correggerla usando le funzioni di edit, o premere **ENTER** per confermarla. Per lasciare il modo NUMBER, premere **ENTER** quando una linea appare senza istruzioni o premere **FCTN 4** (CLEAR) quando viene visualizzata una qualsiasi riga completa. NUMBER puo' essere abbreviato con NUM.

**Opzioni**

Potete specificare una *linea-iniziale* e/o un *incremento*.

**Esempio**

Nell'esempio che segue tutto cio' che deve essere digitato e' SOTTOLINEATO. Premere **ENTER** dopo ciascuna linea.

NUM dice al computer di numerare automaticamente le linee da 100 con incremento 10.

```
>NUM
>100 X=4
>110 Z=10
>120
```

NUM 110 spiega al computer di numerare automaticamente a partire da 110 con incremento di 10. Cambiare la linea 110 in Z=11.

```
>NUM 110
>110 Z=11
>120 PRINT (Y+X)/Z
>130
```

NUM 105,5 dice al computer di numerare automaticamente a partire da 105 con incremento di 5. La linea 110 esiste gia'.

```
>NUM 105,5
>105 Y=7
>110 Z=11
>115
>LIST
>100 X=4
>105 Y=7
>110 Z=11
>120 PRINT (Y+X)/Z
```

---

# OLD

---

**Formato**

OLD ["] *periferica-nome-programma* ["]

**Descrizione**

Il comando OLD carica in memoria il *programma* dalla *periferica* specificata. Il programma deve essere prima stato salvato sulla *periferica* specificata usando il comando SAVE. OLD chiude tutti i files aperti e cancella il programma attualmente in memoria prima di caricare il nuovo *programma*. Per aggiungere linee di programma da un altro programma al programma in memoria, vedere il comando MERGE.

La *periferica* puo' indicare piu' di un dispositivo di memoria di massa. Se e' CS1 o CS2, indica uno dei due possibili registratori a cassetta, e non si puo' specificare il *nome-programma*. Il programma caricato e' quello che si trova sulla cassetta. Le istruzioni su come operare con il registratore sono visualizzate sullo schermo.

Vedere il manuale del sistema di memoria a dischi per le istruzioni riguardanti l'uso di OLD con i dischetti.

**Esempi**

| | |
|---|---|
| OLD CS1 carica un programma dal registratore a cassetta nella memoria del computer. | >OLD CS1 |
| OLD "DSK1.MYPROG" carica il programma MYPROG nella memoria del computer dal dischetto sul disk drive 1. | >OLD "DSK1.MYPROG" |
| OLD DSK.DISK3.UPDATE80 carica dal dischetto chiamato DISK3 il programma UPDATE80 nella memoria del computer. | >OLD DSK.DISK3.UPDATE80 |

## ON BREAK

**Formato**

ON BREAK STOP
ON BREAK NEXT

**Descrizione**

L'istruzione ON BREAK determina l'azione da prendere se viene incontrato un breakpoint durante l'esecuzione di un programma. L'azione di default e' STOP, che causa l'arresto dell'esecuzione del programma e la stampa del messaggio standard di breakpoint. NEXT e' l'alternativa che trasferisce il controllo alla linea successiva senza che avvenga alcun breakpoint.

Potete usare ON BREAK NEXT per avere un programma che ignori i breakpoints che vi avete inserito per la messa a punto. (NOTA: ON BREAK NEXT non ha alcun effetto su un'istruzione BREAK che non e' seguita da un numero di linea. Il breakpoint avverra' anche se e' stata eseguita l'istruzione ON BREAK NEXT). Quando si inserisce ON BREAK NEXT in un programma, il tentativo di arrestarlo mediante **FCTN 4** (CLEAR) non ha alcun effetto. In questo caso solo **FCTN +** (QUIT) puo' fermare il programma. **FCTN +** (QUIT) cancella il programma e riporta alla maschera d'accensione e puo' interferire sul regolare funzionamento di periferiche esterne come i disk drives.

**Programma**

Il programma sulla destra spiega l'uso di ON BREAK. La linea 110 pone un breakpoint alla linea 150. La linea 120 annulla gli effetti dei breakpoints trasferendo il controllo alla linea successiva. Un breakpoint avviene alla linea 130 nonostante la linea 120. Inserire CONTINUE. Non ci sara' arresto alla linea 150 per effetto della linea 120. **FCTN 4** (CLEAR) non ha alcun effetto durante l'esecuzione tra le linee 140 e 160 per effetto della linea 120. La linea 170 ripristina l'uso normale di **FCTN 4** (CLEAR).

```
>100 CALL CLEAR
>110 BREAK 150
>120 ON BREAK NEXT
>130 BREAK
>140 FOR A=1 TO 50
>150 PRINT "FCTN 4 E' DISABIL
 ITATO"
>160 NEXT A
>170 ON BREAK STOP
>180 FOR A=1 TO 50
>190 PRINT "ADESSO FUNZIONA"
>200 NEXT A
```

---

# ON ERROR

---

**Formato**

ON ERROR STOP
ON ERROR *numero-linea*

**Descrizione**

L'istruzione ON ERROR determina l'azione da eseguire se avviene un errore durante l'esecuzione di un programma. L'azione di default e' STOP, che provoca la stampa del messaggio standard di errore e l'arresto del programma. L'alternativa e' data dall'indicazione del *numero-linea*, il che trasferisce il controllo a quella linea in caso di errore.

Una volta che e' avvenuto un errore ed il controllo e' stato trasferito, il funzionamento in caso di errore e' ripristinato al modo normale, cioe' con STOP. Se desiderate che siano trattati diversamente altri errori deve essere eseguita nuovamente l'istruzione ON ERROR.

Se e' specificato un *numero-linea* da ON ERROR, il *numero-linea* deve essere l'inizio di una subroutine similarmente a quella chiamata da GOSUB. Essa deve terminare con un'istruzione RETURN. Vedere RETURN (con ON ERROR) per maggiori informazioni.

NOTA: un trasferimento del controllo successivo all'esecuzione di un'istruzione ON ERROR funziona come l'esecuzione di un'istruzione GOSUB. Come con GOTO e GOSUB, e' meglio evitare trasferimenti da e verso sottoprogrammi. Il risultato piu' comune di un trasferimento illegale all'interno di un sottoprogramma e' un *SYNTAX ERROR* su un'istruzione che sembra essere corretta.

**Programma**

Il programma sulla destra illustra l'uso di ON ERROR. La linea 110 fa in modo che il controllo sia passato alla linea 160 in caso di errore.

La linea 170 fa in modo che in caso di un successivo errore il controllo passi alla linea 230.
La linea 180 verifica il tipo di errore tramite CALL ERR.
La linea 190 trasferisce il controllo alla linea 230 se l'errore non e' del tipo previsto.
La linea 210 cambia il valore di X$ in un valore accettabile.
La linea 220 riporta il controllo alla linea nella quale era avvenuto l'errore.
La linea 240 riporta la natura dell'errore imprevisto ed il programma si ferma.

```
>100 CALL CLEAR
>110 ON ERROR 160
>120 X$="A"
>130 X=VAL(X$)
>140 PRINT X;"AL QUADRATO E'"
 ;X*X
>150 STOP
>160 REM SUBROUTINE ERRORE
>170 ON ERROR 230
>180 CALL ERR(CODICE,TIPO,GRA
 VITA,LINEA)

>190 IF LINEA<>130 THEN RETUR
 N 230
>200 IF CODICE<>74 THEN RETUR
 N 230
>210 X$="5"
>220 RETURN
>230 REM ERRORE SCONOSCIUTO

>240 PRINT "ERRORE";CODICE;"
 NELLA LINEA";LINEA
>RUN
  5 AL QUADRATO E' 25
```

# ON GOSUB

### Formato

ON *espressione-numerica* GOSUB *numero-linea* [,...]
ON *espressione-numerica* GO SUB *numero-linea* [,...]

### Descrizione

L'istruzione ON GOSUB trasferisce il controllo alla subroutine che inizia con il *numero-linea* specificato a seconda del valore dell'*espressione-numerica*. Oltre ad offrire una scelta essa funziona allo stesso modo dell'istruzione GOSUB ma e' piu' efficiente in quanto richiede meno linee di quante ne occorrerebbero con l'istruzione IF-THEN-ELSE. ON...GOSUB non puo' essere usata per trasferire il controllo ad o da un sottoprogramma.

L'*espressione-numerica* deve avere un valore da 1 fino al totale dei *numeri di linea*.

### Esempi

ON X GOSUB 1000,2000,300 trasferisce il controllo alla linea 1000 se X e' 1, alla 2000 se X e' 2 e alla 300 se X e' 3.

```
>100 ON X GOSUB 1000,2000,300
```

ON P-4 GOSUB 200,250,300,800,170 trasferisce il controllo alla linea 200 se P-4 e' uguale a 1 (P e' 5), alla 250 se P-4 e' 2, alla 300 se P-4 e' 3, alla 800 se P-4 e' 4 ed alla 170 se P-4 e' 5.

```
>100 ON P-4 GOSUB 200,250,300
,800,170
```

**Programma**

Il programma a destra illustra un uso di ON...GOSUB. La linea 220 determina dove saltare in funzione del valore di CHOICE.

```
>100 CALL CLEAR
>110 DISPLAY AT(11,1):"SCEGLI
 UNO DEI SEGUENTI:"
>120 DISPLAY AT(13,1):"1 ADDI
 ZIONA DUE NUMERI."
>130 DISPLAY AT(14,1):"2 MOLT
 IPLICA DUE NUMERI."
>140 DISPLAY AT(15,1):"3 SOTT
 RAI DUE NUMERI."
>150 DISPLAY AT(20,1):"SCELTA
 ?:"
>160 DISPLAY AT(22,2):"PRIMO
 NUMERO:"
>170 DISPLAY AT(23,1):"SECOND
 O NUMERO:"
>180 ACCEPT AT(20,14)VALIDATE
 (NUMERIC):CHOICE
>190 IF CHOICE<1 OR CHOICE>3
 THEN 180
>200 ACCEPT AT(22,16)VALIDATE
 (NUMERIC):PRIMO
>210 ACCEPT AT(23,16)VALIDATE
 (NUMERIC):SECONDO
>220 ON CHOICE GOSUB 240,260,
 280
>230 GOTO 180
>240 DISPLAY AT(3,1):PRIMO;"P
 IU´";SECONDO;"UGUALE";PRIMO+
 SECONDO
>250 RETURN
>260 DISPLAY AT(3,1):PRIMO;"P
 ER";SECONDO;"UGUALE";PRIMO*S
 ECONDO
>270 RETURN
>280 DISPLAY AT(3,1):PRIMO;"M
 ENO";SECONDO;"UGUALE";PRIMO-
 SECONDO
>290 RETURN
```

(Premere **FCTN 4** per fermare il programma.)

---

# ON GOTO

---

**Formato**

ON *espressione-numerica* GOTO *numero-linea* [,...]
ON *espressione-numerica* GO TO *numero-linea* [,...]

**Descrizione**

L'istruzione ON GOTO trasferisce il controllo al *numero-linea* nella posizione corrispondente al valore dell'*espressione-numerica*. Oltre che a offrire una scelta essa funziona come l'istruzione GOTO, ma e' piu' efficiente in quanto richiede meno linee dell'uso di IF-THEN-ELSE.ON...GOTO non puo' essere usata per trasferire il controllo da o ad un sottoprogramma.

L'*espressione-numerica* deve avere un valore compreso tra 1 ed il totale dei *numeri di linea*.

**Esempi**

ON X GOTO 1000,2000,300 trasferisce il controllo alla linea 1000 se X e' 1, alla 2000 se X e' 2 e alla 300 se X e' 3.

```
>100 ON X GOTO 1000,2000,300
```

ON P-4 GOTO 200,250,300,800,170 trasferisce il controllo alla linea 200 se P-4 e' uguale a 1, (P e' 5), alla 250 se P-4 e' 2, alla 300 se P-4 e' 3, alla 800 se P-4 e' 4 ed alla 170 se P-4 e' 5.

```
>100 ON P-4 GOTO 200,250,300,
 800,170
```

**Programma**

Il programma a destra illustra un uso di ON...GOTO. La linea 220 determina dove saltare in funzione del valore di CHOICE.

```
>100 CALL CLEAR
>110 DISPLAY AT(11,1):"SCEGLI
 UNO DEI SEGUENTI:"
>120 DISPLAY AT(13,1):"1 ADDI
 ZIONA DUE NUMERI."
>130 DISPLAY AT(14,1):"2 MOLT
 IPLICA DUE NUMERI."
>140 DISPLAY AT(15,1):"3 SOTT
 RAI DUE NUMERI."
>150 DISPLAY AT(20,1):"SCELTA
 ?:"
>160 DISPLAY AT(22,2):"PRIMO
 NUMERO:"
>170 DISPLAY AT(23,1):"SECOND
 O NUMERO:"
>180 ACCEPT AT(20,14)VALIDATE
 (NUMERIC):CHOICE
>190 IF CHOICE<1 OR CHOICE>3
 THEN 180
>200 ACCEPT AT(22,16)VALIDATE
 (NUMERIC):PRIMO
>210 ACCEPT AT(23,16)VALIDATE
 (NUMERIC):SECONDO
>220 ON CHOICE GOTO 230,250,2
 70
>230 DISPLAY AT(3,1):PRIMO;"P
 IU'";SECONDO;"UGUALE";PRIMO+
 SECONDO
>240 GOTO 180
>250 DISPLAY AT(3,1):PRIMO;"P
 ER";SECONDO;"UGUALE";PRIMO*S
 ECONDO
>260 GOTO 180
>270 DISPLAY AT(3,1):PRIMO;"M
 ENO";SECONDO;"UGUALE";PRIMO-
 SECONDO
>280 GOTO 180
```

(Premere **FCTN 4** per fermare il programma.)

---

# ON WARNING

---

**Formato**

ON WARNING PRINT
ON WARNING STOP
ON WARNING NEXT

**Descrizione**

L'istruzione ON WARNING determina l'azione da seguire se durante l'esecuzione del programma avviene un errore lieve (con visualizzazione del relativo messaggio di avviso). L'azione di default e' PRINT, che provoca la visualizzazione del messaggio ed il proseguimento del programma. Un'alternativa e' STOP, che provoca la visualizzazione del messaggio e l'arresto dell'esecuzione del programma. L'altra alternativa e' NEXT che provoca la continuazione del programma senza che venga visualizzato alcun messaggio d'avviso.

**Programma**

Il programma sulla destra spiega l'uso di ON WARNING. La linea 110 stabilisce che il trattamento di avviso vada alla linea successiva. La linea 120 quindi stampa il risultato senza alcun messaggio.

```
>100 CALL CLEAR
>110 ON WARNING NEXT
>120 PRINT 120,5/0
```

La linea 130 pone il funzionamento nella condizione di default, stampa il messaggio di avviso e continua l'esecuzione. La linea 140 percio' stampa 140, poi il messaggio e quindi continua l'esecuzione del programma.

```
>130 ON WARNING PRINT
>140 PRINT 140,5/0
```

La linea 150 pone il trattamento di avviso in modo che sia stampato il messaggio e poi si arresti l'esecuzione del programma. Quindi stampa 160, il messaggio, e poi il programma si arresta.

```
>150 ON WARNING STOP
>160 PRINT 160,5/0
>170 PRINT 170
>RUN
  120           9.99999E+**
  140
 * WARNING
 NUMERIC OVERFLOW IN 140
                9.99999E+**
  160
 * WARNING
 NUMERIC OVERFLOW IN 160
```

# OPEN

**Formato**

OPEN *#numero-file:periferica-nomefile* [,*organizzazione-file*]
[,*tipo-file*] [,*modo-apertura*] [,*tipo-record*]

**Descrizione**

L'istruzione OPEN prepara un programma BASIC ad usare files di dati registrati su dischetto o cassetta stabilendo un collegamento tra il *numero-file* ed un file. Per disporre questo colegamento, l'istruzione OPEN descrive le caratteristiche del file. Se il file gia' esiste, la descrizione che viene data nel programma deve corrispondere alle attuali caratteristiche del file. Files su cassette non vengono controllati, tuttavia, possono pero' avvenire degli errori se le caratteristiche non corrispondono.
Il *numero-file* deve essere incluso nell'istruzione OPEN. Le istruzioni che si riferiscono ai files hanno bisogno di un *numero-file* che deve essere compreso tra 0 e 255. Il numero di file 0 e' riservato alla tastiera ed allo schermo del computer. Esso non puo' essere usato per altri files ed e' sempre aperto. E' possibile assegnare gli altri numeri a piacimento con un numero per ciascun file. Il *numero-file* viene immesso come un segno di sterlina (#) seguito da un'espressione numerica che, quando arrotondata al numero intero piu' vicino, e' un numero compreso tra 0 e 255 e non deve essere il numero di un file che e' gia' stato aperto.
Nell'istruzione OPEN deve essere inclusa anche la *periferica*. Se essa e' CS1 o CS2, che designa uno dei 2 possibili registratori a cassette, non si puo' specificare il *nomefile*. Le istruzioni su come operare con il registratore a cassette vengono visualizzate sullo schermo.
Se la *periferica* e' DSK1, DSK2 o DSK3 (o anche DSK4 se si utilizza un particolare tipo di disk controller) il *nomefile* e' il nome di un file esistente sul dischetto nel drive specificato. Se la *periferica* e' DSK.*nome-disco*, dove *nome-disco* e' il nome assegnato ad un dischetto presente in uno dei drives, allora il *nomefile* e' il nome di un file sul dischetto indentificato dal *nome-dischetto*. Il computer cerca i drives, a partire dal DSK1, finche' non trova il dischetto con il nome dato. Successivamente esso cerca il *nomefile* su quel dischetto.
Le altre informazioni possono trovarsi in qualsiasi ordine o essere omesse. Se un parametro viene omesso il computer assume per questi i valori di default, che sono qui di seguito descritti.

*Organizzazione-file* puo' essere o sequenziale oppure casuale. I records in un file sequenziale sono scritti o letti uno di seguito all'altro. I records dei files ad accesso casuale (random) possono essere letti o scritti in qualsiasi ordine. I files random possono anche essere elaborati sequenzialmente. Per

indicare quale struttura ha il file, inserire o SEQUENTIAL per i files sequenziali, o RELATIVE per i files random. E' possibile facoltativamente specificare il numero iniziale dei records su un file facendo seguire ai parametri SEQUENTIAL o RELATIVE un'espressione numerica. Se non e' specificata l'*organizzazione-file* il default e' SEQUENTIAL.

Il *tipo-file* puo' essere DISPLAY o INTERNAL. I files possono essere scritti sia in formato leggibile all'uomo, chiamato ASCII (DISPLAY) o in formato comprensibile alla macchina, chiamato binario (INTERNAL). I records binari occupano meno spazio e sono elaborati piu' velocemente dal computer. Tuttavia se le informazioni dovranno essere stampate o visualizzate, il formato ASCII e' normalmente la scelta migliore.

Per specificare che il file deve essere in formato ASCII, assegnare DISPLAY. Per specificare il formato binario assegnare INTERNAL. Se non specificate il *tipo-file* il default e' DISPLAY. Normalmente la scelta migliore e' il formato INTERNAL quando si usano files su cassette o dischetti, e DISPLAY quando si usano files destinati all'uso con la stampante o l'interfaccia RS232.

Il *modo-apertura* puo' essere UPDATE, INPUT, OUTPUT o APPEND. Il computer puo' essere informato del fatto che sul file si puo' sia leggere che scrivere, che esso puo' essere solo letto, che su esso ci si puo' solo scrivere, oppure che vi si possono soltanto aggiungere records. Tuttavia se il file e' stato protetto non ci si puo' scrivere, ma puo' essere soltanto aperto per la lettura.

Per poter sia leggere che scrivere su un file specificare UPDATE. Per poter solo leggere da un file specificare INPUT. Per poter solo scrivere su un file specificare OUTPUT. Se si vuole solamente aggiungere records specificare APPEND. Il modo APPEND puo' essere specificato per i records a lunghezza VARIABLE. Se non e' specificato il *modo-apertura* il default e' UPDATE.

Notare che se esiste gia' un file non protetto su dischetto, lo specificare OUTPUT come *modo-apertura* per lo stesso file, provoca la sovrascrittura del nuovo file sul file esistente. Potete prevenire cio' spostandovi alla fine del file tramite o mediante l'uso dell'istruzione RESTORE con l'appropriato numero di record, o aprendo il file nel modo APPEND.

Il *tipo-record* puo' essere o VARIABLE o FIXED. I files possono avere records tutti della stessa lunghezza o records di lunghezza variabile. Se tutti i records sono della stessa lunghezza quelli che sono piu' corti vengono riempiti per colmare la differenza. Potete specificare i records di lunghezza variabile mediante il parametro VARIABLE. Per specificare records di lunghezza fissa, usare FIXED.

Se desiderate, potete specificare una lunghezza massima per ciascun record facendo seguire a VARIABLE o FIXED un'espressione numerica. La lunghezza massima possibile per ciascun record dipende dalla periferica usata. Se non e' specificata la lunghezza del record, il default e' 80 per i dischetti, 64 per le cassette, 80 per l'interfaccia RS232 e 32 per la stampante termica.

I files RELATIVE devono avere records di lunghezza FIXED. Se non e' specificato il *tipo-record* per un file RELATIVE, il default e' FIXED.

I files SEQUENTIAL possono essere sia FIXED che VARIABLE. Se non specificate un *tipo-record* per un file SEQUENTIAL, il default e' VARIABLE. Un file di lunghezza fissa puo' essere riaperto per altri accessi SEQUENTIAL o RELATIVE indipendentemente da quanto assegnato in precedenza ad *organizzazione-file*.

**Esempi**

OPEN #1:"CS1",FIXED,OUTPUT apre un file sul registratore 1. Il file e' SEQUENTIAL, in formato DISPLAY, in modo OUTPUT con records di lunghezza FIXED fino ad un massimo di 64 bytes.

```
>100 OPEN #1:"CS1",FIXED,OUTP
 UT
```

OPEN #23:"DSK.MYDISK.X",RELATIVE 100,INTERNAL,UPDATE,FIXED apre un file di nome "X". Il file e' sul dischetto chiamato MYDISK che si trova in qualsiasi drive nel quale e' inserito il dischetto. Il file e' RELATIVE, in formato INTERNAL, con records di lunghezza FIXED fino ad un massimo di 80 bytes. Il file e' aperto in modo UPDATE e inizialmente con 100 records disponibili.

```
>300 OPEN #23:"DSK.MYDISK.X",
 RELATIVE 100,INTERNAL,UPDATE
 ,FIXED
```

OPEN #243:A$,INTERNAL, se A$ e' uguale a "DSK2.ABC" il computer assegna un file sul dischetto nel drive 2 con il nome ABC. Il file e' SEQUENTIAL, in formato INTERNAL, modo UPDATE con records di lunghezza VARIABLE fino ad una lunghezza massima di 80 bytes.

```
>100 OPEN #243:A$,INTERNAL
```

OPEN #17:"TP",OUTPUT prepara la stampante termica per la stampa.

```
>100 OPEN #17:"TP",OUTPUT
```

# OPTION BASE

**Formato**

OPTION BASE 0
OPTION BASE 1

**Descrizione**

L'istruzione OPTION BASE pone a 0 o a 1 il piu' basso indice possibile nelle matrici. Il default e' 0. Se e' usata un'istruzione OPTION BASE, essa deve trovarsi ad un numero di linea piu' basso di ciascuna istruzione DIM o di altri riferimenti alle matrici. In un programma ci puo' essere una sola istruzione OPTION BASE, ed essa ha effetto su tutti gli indici delle matrici. L'istruzione OPTION BASE non puo' comparire in un'istruzione IF-THEN-ELSE

**Esempio**

OPTION BASE 1 pone ad 1 l'indice di partenza di tutte le matrici.

```
>100 OPTION BASE 1
```

---

# Sottoprogramma PATTERN

---

**Formato**

CALL PATTERN(#*numero-sprite*,*valore-carattere* [,...])

**Descrizione**

Il sottoprogramma PATTERN permette di cambiare la forma del carattere che costituisce uno sprite senza influenzare le altre caratteristiche di esso.
Il *numero-sprite* specifica lo sprite che si sta usando. Il *valore-carattere* puo' essere un numero intero compreso tra 32 e 143. Vedere il sottoprogramma CHAR per informazioni riguardanti la definizione della forma dei caratteri. Riferirsi al sottoprogramma MAGNIFY per ulteriori informazioni.

**Programma**

Il programma a destra illustra l'uso del sottoprogramma PATTERN. Le linee da 110 a 140 costruiscono un piano.

Le linee dalla 150 alla 200 definiscono i caratteri da 96 a 107.

La linea 210 crea uno sprite con una forma di una ruota che inizia a muoversi verso destra.
La linea 220 da' allo sprite una dimensione doppia.
Le linee da 230 a 270 danno la parvenza del movimento alla ruota dato che il carattere visualizzato e' cambiato.

Vedere anche il terzo esempio del sottoprogramma SPRITE.

```
>100 CALL CLEAR
>110 CALL COLOR(12,16,16)
>120 FOR A=19 TO 24
>130 CALL HCHAR(A,1,120,32)
>140 NEXT A
>150 A$="01071B21214141FFFF41
 41212119070080E09884848282FF
 FF8282848498E000"
>160 B$="0106182030 5C46818142
 46242C18070080601834246242 81
 81623A0C0418E000"
>170 C$="0106182C244642818146
 5C302018070080601804 0C3A6281
 81426224341 8E000"
>180 CALL CHAR(96,A$)
>190 CALL CHAR(100,B$)
>200 CALL CHAR(104,C$)
>210 CALL SPRITE(#1,96,5,130,
 1,0,8)
>220 CALL MAGNIFY(3)
>230 FOR A=96 TO 104 STEP 4
>240 CALL PATTERN(#1,A)
>250 FOR PAUSA=1 TO 5:: NEXT
PAUSA
>260 NEXT A
>270 GOTO 230
```

(Premere **FCTN 4** per fermare il programma).

---

# Sottoprogramma PEEK

---

**Formato**

CALL PEEK(*indirizzo,lista variabili-numeriche*)

**Descrizione**

Il sottoprogramma PEEK viene usato insieme a INIT, LINK e LOAD per accedere ai sottoprogrammi in linguaggio Assembly. Il sottoprogramma PEEK restituisce nelle variabili specificate nelle *lista variabili-numeriche* i valori che corrispondono a quelli contenuti nel byte specificato nell'*indirizzo* ed in quelli seguenti. PEEK puo' essere usato senza sottoprogrammi in linguaggio Assembly, ma le informazioni ottenute sono di poca utilita'.

Istruzioni dettagliate sull'uso di INIT, LINK, LOAD e PEEK sono incluse nei programmi scritti che sono disponibili su cassette o su dischetti.

Un uso indiscriminato di PEEK puo' provocare il blocco ("lock up") che richiede lo spegnimento del computer prima di poterlo riutilizzare.

**Esempi**

CALL PEEK(8192,X1,X2,X3,X4) riporta nelle variabili X1, X2, X3 e X4 i valori contenuti nelle locazioni 8192, 8193, 8194 e 8195.

```
>100 CALL PEEK(8192,X1,X2,X3,
X4)
```

---

# PI

---

**Formato**

PI

**Descrizione**

La funzione PI restituisce il valore del pi greco che e' uguale a 3.14159265359

**Esempio**

VOLUME=4/3*PI*6 3 assegna alla variabile VOLUME i quattro terzi di pi greco moltiplicato 6 elevato al cubo, il che e' il volume di una sfera con raggio 6.

```
>100 VOLUME=4/3*PI*6 3
```

# POS

==========================================================================

**Formato**

POS(*stringa 1*,*stringa 2*,*espressione-numerica*)

**Descrizione**

La funzione POS restituisce la posizione di *stringa 2* la prima volta che viene trovato in *stringa 1*. La ricerca inizia dalla posizione specificata dall'*espressione-numerica*. Se la ricerca non ha esito la funzione restituisce il valore zero.

**Esempi**

X=POS("PAN","A",1) assegna ad X il valore 2 poiche' A e' la seconda lettera della stringa PAN.

```
>100 X=POS("PAN","A",1)
```

Y=POS("APAN","A",2) pone Y uguale a 3 poiche' A e' nella terza posizione nella stringa APAN ed e' la prima volta che A viene individuata nella porzione di APAN nella quale e' stata eseguita la ricerca.

```
>100 Y=POS("APAN","A",2)
```

Z=POS("PAN","A",3) pone Z uguale a 0 poiche' A non si trova nella parte controllata di PAN.

```
>100 Z=POS("PAN","A",3)
```

R=POS("PABNAN","AN",1) pone R uguale a 5 poiche' la prima volta che si incontra AN inizia con la A nella quinta posizione di PABNAN.

```
>100 R=POS("PABNAN","AN",1)
```

**Programma**

Il programma a destra illustra un uso di POS. In esso dopo ciascun INPUT vengono ricercati gli spazi, quindi viene stampata una parola per ogni riga.

```
>100 CALL CLEAR
>110 PRINT "IMMETTI UNA FRAS
 SE"
>120 INPUT X$
>130 S=POS(X$," ",1)
>140 IF S=0 THEN PRINT X$ &:
 PRINT :: GOTO 110
>150 Y$=SEG$(X$,1,S):: PRINT
 Y$
>160 X$=SEG$(X$,S+1,LEN(X$))
>170 GOTO 130
```

(Premere **FCTN 4** per fermare il programma).

---

# Sottoprogramma POSITION

---

**Formato**

CALL POSITION(*#numero-sprite,pixel-riga,pixel-colonna* [,...])

**Descrizione**

Il sottoprogramma POSITION riporta la posizione dello sprite o degli srites specificati nei *pixel-riga* e *pixel-colonna* dati come numeri da 1 a 256. Essi sono la posizione dell'angolo superiore sinistro dello sprite. Se lo sprite non e' definito, il *pixel-riga* e il *pixel-colonna* sono posti a zero.

Lo sprite continua a muoversi dopo che e' stata restituita la sua posizione, e quindi bisogna tenerne conto. La distanza coperta dipende dalla velocita' dello sprite.

**Esempio**

CALL POSITION(#1,Y,X) riporta la posizione dell'angolo superiore sinistro dello sprite #1.

```
>100 CALL POSITION(#1,Y,X)
```

Vedere anche il terzo esempio del sottoprogramma SPRITE.

# PRINT

==================================================================

**Formato**

PRINT [#*numero-file* [,REC *numero-record*]:] [*lista-stampa*]

**Descrizione**

L'istruzione PRINT permette di trasferire i valori degli elementi specificati opzionalmente nella *lista-stampa* sullo schermo o facoltativamente su un file o su altra periferica come la stampante. La *lista-stampa* consiste in costanti di stringa o numeriche, variabili di stringa o numeriche, espressioni numeriche o di stringa, e/o la funzione TAB. Ciascun elemento della *lista-stampa* deve essere separato dagli altri mediante una virgola, un punto e virgola o un due punti.
I due punti, la virgola ed il punto e virgola controllano lo spazio sullo schermo o su un file aperto in formato DISPLAY. Con il punto e virgola l'elemento successivo sara' posto subito dopo il precedente. Una virgola fa in modo che l'elemento seguente della *lista-stampa* sia posto nel campo successivo di stampa. Ciascun campo di stampa e' lungo 14 caratteri. Il numero dei campi di stampa dipende dalla lunghezza del record e a seconda del dispositivo che si sta usando. Sullo schermo i campi di stampa sono alla posizione 1 e 15. Se il cursore ha superato l'inizio dell'ultimo campo di stampa, la voce successiva viene stampata sulla riga seguente. I due punti fanno in modo che l'elemento successivo venga posto sulla riga o sul record che segue. Per stampare varie linee vuote, si possono mettere tanti due punti, dopo l'istruzione PRINT, quante sono le righe vuote desiderate. Tuttavia i due punti devono essere separati tra loro da almeno uno spazio, per distinguerli dall'elemento separatore d'istruzione (::).
Un elemento separatore puo' essere posto di seguito all'ultimo elemento della *lista-stampa*, il che ha effetto sul posizionamento del successivo elemento della prossima istruzione PRINT, PRINT...USING, DISPLAY (senza AT) o DISPLAY...USING (senza AT) se riferite alla medesima periferica. Cio' fa in modo che la successiva istruzione di output sia considerata come una continuazione di quella in corso a meno che essa non sia un'istruzione PRINT con la clausola REC.
Mentre si sta stampando una nuova riga sullo schermo, ogni cosa (eccetto gli sprites) gia' presente viene fatta scorrere di una riga verso l'alto (cosicche' viene persa la prima linea in alto) e la nuova riga e' stampata sull'ultima riga in basso dello schermo.

**Opzioni**

Il *#numero-file* determina il file sul quale bisogna stampare. Se omesso o specificando #0, viene considerato lo schermo. Altrimenti il *numero-file* deve essere il numero di un file che e' stato gia' aperto. Vedere OPEN.

La clausola REC e' usata per specificare il record sul quale desiderate stampare gli elementi della *lista-stampa*. REC puo' essere usato soltanto con i files aperti in modo RELATIVE. Vedere OPEN.
Nella scrittura dei files in formato INTERNAL, sia la virgola che il punto e virgola collocano gli elementi della *lista-stampa* adiacenti uno con l'altro. Nei files in formato DISPLAY la virgola ed il punto e virgola si comportano come descritto sopra, con il punto e virgola che colloca l'elemento subito dopo il precedente e la virgola che lo pone nel successivo campo di stampa.

**Esempi**

| | |
|---|---|
| PRINT fa in modo che una linea vuota compaia sul video. | >100 PRINT |
| PRINT "LA RISPOSTA E'";RISP fa si che la costante di stringa LA RISPOSTA E' sia stampata sul video, immediatamente seguita dal valore della variabile RISP. Se RISP e' positivo, ci sara' uno spazio vuoto per il segno positivo dopo "E'". | >100 PRINT "LA RISPOSTA E'";R ISP |
| PRINT X:Y/2 stampa il valore di X su una linea e il valore di Y/2 su quella sottostante. | >100 PRINT X:Y/2 |
| PRINT #12,REC 7:A scrive il valore di A sull'ottavo record del file che era stato aperto con #12 (Il primo record e' il record numero 0). | >100 PRINT #12,REC 7:A |
| PRINT #32:A,B,C, scrive i valori di A, B e C sul prossimo record del file che era stato aperto con il numero 32. La virgola finale crea una condizione di ingresso in sospeso. L'istruzione PRINT successiva, diretta al file #32, scrivera' sullo stesso record di questa istruzione PRINT a meno che essa non specifichi un record, chiudendo quindi la condizione di ingresso in sospeso. | >100 PRINT #32:A,B,C, |

PRINT #1,REC 3:A,B seguito da PRINT #1:C,D fa si che A e B siano scritti sul record 3 del file aperto con #1; C e D saranno scritti sul record 4 dello stesso file.

```
>100 PRINT #1,REC 3:A,B
>150 PRINT #1:C,D
```

**Programma**

Il programma a destra stampa vari valori in diverse posizioni del video.

```
>100 CALL CLEAR
>110 PRINT 1;2;3;4;5;6;7;8;9
>120 PRINT 1,2,3,4,5,6
>130 PRINT 1:2:3
>140 PRINT
>150 PRINT 1;2;3;
>160 PRINT 4;5;6/4
>RUN
 1  2  3  4  5  6  7  8  9
 1              2
 3              4
 5              6
 1
 2
 3

 1  2  3  4  5  1.5
```

----------------------------------------------------------------

# PRINT USING

================================================================

**Formato**

PRINT [#*numero-file*[,REC *numero-record*],]USING *espres.stringa*
        *:lista-stampa*
PRINT [#*numero-file*[,REC *numero-record*],]USING *numero-linea*
        *:lista-stampa*

**Descrizione**

L'istruzione PRINT...USING funziona allo stesso modo di PRINT con l'aggiunta della clausola USING, la quale specifica il formato da usare. L'*espressione-stringa* definisce il formato nella maniera descritta in IMAGE. Il *numero-linea* si riferisce al numero di linea di un'istruzione IMAGE. Vedere l'istruzione IMAGE per maggiori informazioni sull'uso dell'*espressione-stringa*.

**Esempi**

| | |
|---|---|
| PRINT USING "###.##":32.5 stampa 32.50 . | >100 PRINT USING "###.##":32.5 |
| PRINT USING "LA RISPOSTA E' ###.#":123.98 stampa LA RISPOSTA E' 124.0 . | >100 PRINT USING "LA RISPOSTA E' ###.#":123.98 |
| PRINT USING 185:37.4,-86.2 stampa i valori di 37.4 e - 86.2 usando l'istruzione IMAGE della linea 185. | >100 PRINT USING 185:37.4,-86.2 |

---

# RANDOMIZE

---

### Formato

RANDOMIZE [*espressione-numerica*]

### Descrizione

L'istruzione RANDOMIZE resetta la generazione dei numeri casuali in una sequenza imprevedibile. Se RANDOMIZE e' seguita da un'*espressione-numerica*, viene prodotta la stessa sequenza casuale di numeri ogni qualvolta viene eseguita l'istruzione con quel valore. Valori diversi danno sequenze diverse.

### Programma

Il programma a destra illustra un uso dell'istruzione RANDOMIZE. Esso accetta un valore per l'espressione numerica e stampa i primi 10 valori ottenuti usando la funzione RND.

```
>100 CALL CLEAR
>110 INPUT "SEME: ":S
>120 RANDOMIZE S
>130 FOR A=1 TO 10::PRINT A;R
 ND::NEXT A::PRINT
>140 GOTO 110
```

(Premere **FCTN 4** per fermare il programma).

---

# READ

---

**Formato**

READ *lista-variabili*

**Descrizione**

L'istruzione READ permette di assegnare costanti numeriche e di stringa da un'istruzione DATA alle variabili specificate nella *lista-variabili*. La *lista-variabili* consiste di variabili numeriche e di stringa separate da virgole.

I dati sono normalmente letti a partire dalla prima istruzione DATA del programma. Dopo che i dati sono stati letti, il computer memorizza il punto in cui e' finita la lettura e continua da quel punto quando sara' eseguita la successiva istruzione READ. E' possibile cambiare l'ordine di lettura dei dati utilizzando l'istruzione RESTORE.

Vedere gli esempi dell'istruzione DATA.

---

# REC

---

**Formato**

REC (*numero-file*)

**Descrizione**

La funzione REC riporta il numero del record al quale accedure con la successiva PRINT, INPUT o LINPUT nel file aperto con il *numero-file*. I records in un file sono numerati a partire da 0, cosi' il record numero 3, per esempio, e' il quarto record di un file.

**Esempio**

PRINT REC(4) stampa la posizione del record corrente del file aperto con il numero 4.

```
>100 PRINT REC(4)
```

**Programma**

Il programma a destra illustra un uso della funzione REC. La linea 110 apre un file.

Le linee da 120 a 140 scrivono 4 records sul file.

La linea 150 riposiziona il file all'inizio.

Le linee da 160 a 200 stampano la posizione del file, leggono e stampano i valori in quella posizione.

La linea 210 chiude il file.

```
>100 CALL CLEAR
>110 OPEN #1:"DSK1.RNDFILE",
 RELATIVE
>120 FOR A=0 TO 3
>130 PRINT #1:"QUESTO E' IL R
 ECORD",A
>140 NEXT A
>150 RESTORE #1
>160 FOR A=0 TO 3
>170 PRINT REC(1)
>180 INPUT #1:A$,B
>190 PRINT A$;B
>200 NEXT A
>210 CLOSE #1
>RUN
 0
 QUESTO E' IL RECORD 0
 1
 QUESTO E' IL RECORD 1
 2
 QUESTO E' IL RECORD 2
 3
 QUESTO E' IL RECORD 3
```

---

# REM

---

**Formato**

REM *commento*

**Descrizione**

L'istruzione REM permette di inserire commenti in un programma. I commenti possono essere qualsiasi cosa si desideri, ma usualmente sono pero' usati per dividere sezioni dei programmi e per spiegare il significato delle sezioni sottostanti. Non ha importanza cio' che segue REM, compreso il simbolo separatore (::), poiche' i commenti non sono eseguiti e non hanno alcun effetto sul funzionamento del programma. Naturalmente, pero', essi occupano spazio in memoria.

**Esempio**

REM INIZIO SUBROUTINE identifica la sezione in cui ha inizio una subroutine.

```
>100 REM INIZIO SUBROUTINE
```

---

# RESEQUENCE

---

**Formato**

RESEQUENCE [*linea-iniziale*] [,*incremento*]
RES [*linea-iniziale*] [,*incremento*]

**Descrizione**

Il comando RESEQUENCE cambia i numeri di linea del programma in memoria. Se la *linea-iniziale* non viene specificata la numerazione delle linee parte da 100. Se non viene dato l'*incremento*, viene usato un incremento di 10. RESEQUENCE puo' essere abbreviato in RES.

In aggiunta alla rinumerazione delle linee vengono cambiati anche i rinvii che compaiono nelle istruzioni BREAK, DISPLAY...USING, GOSUB, GOTO, IF-THEN-ELSE, ON ERROR, ON...GOSUB, ON...GOTO, PRINT...USING, RESTORE, RETURN e RUN in modo che i riferimenti siano alle stesse linee di codice come prima della rinumerazione. Se in una linea compare un'istruzione che rinvia ad una linea inesistente il numero di linea viene sostituito con 32767.

Se, a causa della *linea-iniziale* e dell'*incremento* scelto, il programma richiede numeri di linea maggiori di 32767, l'operazione di rinumerazione si ferma e la numerazione del programma rimane immutata.

**Esempi**

RES rinumera le linee del programma in memoria a partire da 100 con incremento di 10.

```
>100 RES
```

RES 1000 rinumera le linee del programma in memoria a partire da 1000 con incremento di 10.

```
>100 RES 1000
```

RES 1000,15 rinumera le linee del programma in memoria a partire da 1000 con incremento di 15.

```
>100 RES 1000,15
```

RES ,15 rinumera le linee del programma in memoria a partire da 100 con incremento di 15.

```
>100 RES ,15
```

---

# RESTORE

===

**Formato**

RESTORE [*numero-linea*]
RESTORE #*numero-file* [,REC *numero-record*]

**Descrizione**

L'istruzione RESTORE puo' essere usata sia con le istruzioni DATA che con i files. Quando e' usata con le istruzioni DATA, RESTORE riguarda l'istruzione DATA che sara' usata dalla prossima istruzione READ. Se non e' dato il *numero-linea*, l'istruzione READ sara' usata con la prima istruzione DATA che verra' incontrata. Se e' dato il *numero-linea*, viene usata l'istruzione DATA con quel numero di linea o (se non c'e' l'istruzione DATA) con la successiva istruzione DATA.
Quando e' usata con i files l'istruzione RESTORE riguarda il record usato dalla prossima istruzione PRINT, INPUT o LINPUT che si riferisce ad un *numero-file*. Se non viene specificata la clausola REC, il prossimo record e' il primo record del file, record numero 0. Se e' presente la clausola REC, il *numero-record* specifica il prossimo record da usare.
Se esiste una condizione di ingresso in sospeso per effetto di una precedente PRINT, DISPLAY, PRINT...USING o DISPLAY...USING, allora quel record in sospeso viene scritto sul file prima che sia eseguita l'istruzione RESTORE. La condizione di ingresso in sospeso dei dati viene rimossa dall'istruzione RESTORE.

**Esempi**

| | |
|---|---|
| RESTORE assegna la successiva istruzione DATA che deve essere usata alla prima istruzione DATA del programma. | >100 RESTORE |
| RESTORE 130 assegna la prossima istruzione DATA da usare alla istruzione DATA della linea 130 o, se non vi compare alcun DATA, alla prima istruzione DATA che compare dopo la linea 130. | >100 RESTORE 130 |
| RESTORE #1 assegna il prossimo record da usare mediante la successiva istruzione PRINT, INPUT o LINPUT usando il file #1 a partire dal primo record. | >100 RESTORE #1 |
| RESTORE #4,REC H5 pone il prossimo record da usare mediante l'istruzione PRINT, INPUT o LINPUT, del file #4 a partire dal record H5. | >100 RESTORE #4,REC H5 |

# RETURN (con GOSUB)

**Formato**

RETURN

**Descrizione**

Vedere anche RETURN (con ON ERROR).

RETURN usato con GOSUB riporta il controllo all'istruzione successiva al GOSUB o ON...GOSUB che era stata eseguita per ultima.

**Programma**

Il programma a destra illustra un uso di RETURN quando e' usato con GOSUB. Il programma calcola l'interesse fruttato dà una somma di denaro.

```
>100 CALL CLEAR
>110 INPUT "SOMMA DEPOSITATA:
  ":AMOUNT
>120 INPUT "INTERESSE ANNUALE
  : ":RATE
>130 IF RATE<1 THEN RATE=RATE
 0
>140 PRINT "CALCOLO INTERESSI
  PER VOLTE "
>150 INPUT "ANNUALMENTE: ":CO
 MP
>160 INPUT "A PARTIRE DALL'AN
 NO: ":Y
>170 INPUT "NUMERO DEGLI ANNI
  : ":N
>180 CALL CLEAR
>190 FOR A=Y TO Y+N
>200 GOSUB 240
>210 PRINT A,INT(AMOUNTO+.
 5)/100
>220 NEXT A
>230 STOP
>240 FOR B=1 TO COMP
>250 AMOUNT=AMOUNT+AMOUNT*RAT
 E/(COMPO)
>260 NEXT B
>270 RETURN
```

# RETURN (con ON ERROR)

**Formato**

RETURN [*numero-linea*]
RETURN [NEXT]

**Descrizione**

Vedere anche RETURN (con GOSUB).

RETURN e' usato con ON ERROR. Dopo che e' stata eseguita un'istruzione ON ERROR, un'errore provoca il trasferimento alla linea specificata nell'istruzione ON ERROR. Quella linea, o una dopo di essa, puo' essere un'istruzione RETURN. Se RETURN e' data senza nessuna specificazione dopo di essa, il controllo e' riportato all'istruzione sulla quale era avvenuto l'errore ed il programma la esegue di nuovo. Se RETURN e' seguito dal *numero-linea*, il controllo viene trasferito alla linea specificata e l'esecuzione riprende da quella linea. Se RETURN e' seguito da NEXT, il controllo e' trasferito all'istruzione successiva a quella che ha causato l'errore.

**Programma**

Il programma a destra illustra l'uso di RETURN con ON ERROR. La linea 120 in caso di errore trasferisce il controllo alla linea 170. La linea 130 provoca un errore. La linea 140, seguente a quella che ha causato l'errore, stampa 140. La linea 170 verifica se l'errore e' avvenuto 4 volte ed in caso positivo trasferisce il controllo alla linea 220. La linea 180 incrementa di 1 il contatore di errore. La linea 190 stampa 190. La linea 200 riporta il controllo degli errori alla linea 170. La linea 210 rimanda alla linea che ha causato l'errore e la riesegue. La linea 220, la quale e' stata eseguita soltanto dopo che l'errore e' avvenuto 4 volte, stampa 220 e rimanda alla linea seguente quella che ha causato l'errore.

```
>100 CALL CLEAR
>110 A=1
>120 ON ERROR 170
>130 X=VAL("D")
>140 PRINT 140
>150 STOP
>160 REM CONTROLLO ERRORE
>170 IF A>4 THEN 220
>180 A=A+1
>190 PRINT 190
>200 ON ERROR 170
>210 RETURN
>220 PRINT 220 :: RETURN NEXT
>RUN
  190
  190
  190
  190
  220
  140
```

Vedere anche l'esempio dell'istruzione ON ERROR.

---

# RND

===

**Formato**

RND

**Descrizione**

La funzione RND restituisce il successivo numero pseudo-casuale nella sequenza dei numeri pseudo casuali in corso. Il numero riportato e' maggiore o uguale a 0 e minore di 1. La sequenza di numeri casuali restituita e' la stessa ogni volta che un programma viene eseguito a meno che non appaia nel programma un'istruzione RANDOMIZE.

**Esempi**

COLOR16=INT(RND)+1 assegna a COLOR16 un numero compreso tra 1 e 16.

```
>100 COLOR16=INT(RND)+1
```

VALORE=INT(RND)+10 assegna a VALORE un numero compreso tra 10 e 25.

```
>100 VALORE=INT(RND)+10
```

LL(8)=INT(RND*(B-A+1))+A pone LL(8) uguale ad un numero qualsiasi compreso tra A e B.

```
>100 LL(8)=INT(RND*(B-A+1))+
A
```

---

# RPT$

---

**Formato**

RPT$(*espressione-stringa,espressione-numerica*)

**Descrizione**

La funzione RPT$ restituisce una stringa uguale alla *espressione-stringa* ripetuta tante volte quante specificate nell'*espressione-numerica*. Se RPT$ produce una stringa piu' lunga di 255 caratteri, i caratteri eccedenti vengono troncati e viene dato un messaggio d'avviso.

**Esempi**

M$=RPT$("ABCD",4) pone in M$ "ABCDABCDABCDABCD".

```
>100 M$=RPT$("ABCD",4)
```

CALL CHAR(96,RPT$("0000FFFF",8)) definisce i caratteri da 96 a 99 con la stringa "0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF0000FFFF".

```
>100 CALL CHAR(96,RPT$("0000F
 FFF",8))
```

PRINT USING:RPT$("#",40):X$ stampa il valore di X$ usando una immagine di 40 segni di sterlina.

```
>100 PRINT USING:RPT$("#",40)
 :X$
```

---

# RUN

===

**Formato**

RUN ["*periferica.nome-programma*"]
RUN [*numero-linea*]

**Descrizione**

Il comando RUN, che puo' anche essere usato come istruzione, da' inizio all'esecuzione del programma. Il programma da lanciare dovra' essere prima caricato in memoria specificando la *periferica.nome-programma*. Il computer controlla se ci sono errori nei cicli FOR-NEXT per vedere se sono stati omessi dei NEXT, e se ci sono errori di sintassi. I valori di tutte le variabili numeriche sono riportati a zero e vengono annullati tutti i valori delle variabili stringa (cioe' stringhe vuote). Solo dopo aver eseguito questi controlli il programma inizia la sua esecuzione.

**Opzioni**

Se e' stata specificata la *periferica.nome-programma*, il programma da eseguire viene caricato dal dispositivo specificato. I dati ed i programmi al momento presenti in memoria vengono perduti.
Se si specifica il *numero-linea*, inizia l'esecuzione del programma in memoria a partire da quel *numero-linea*.

**Esempi**

| | |
|---|---|
| RUN fa si che il computer inizi ad eseguire il programma in memoria. | >RUN |
| RUN 200 fa in modo che l'esecuzione del programma inizi dalla linea 200. | >RUN 200<br>>100 RUN 200 |
| RUN "DSK1.PRG3" fa in modo che il computer carichi dal drive 1 il programma chiamato PRG3 ed inizi ad eseguirlo. | >RUN "DSK1.PRG3"<br>>320 RUN "DSK1.PRG3" |

**Programma**

Il programma sulla destra illustra l'uso del comando RUN usato come istruzione. Esso crea un "MENU'" e da' la possibilita' all'utente del programma di scegliere quale programma desidera lanciare.
Gli altri programmi invece di terminare nel modo usuale, dovranno lanciare a loro volta questo dell'esempio per poter riutilizzare il menu'.

```
>100 CALL CLEAR
>110 PRINT "1 PROGRAMMA 1."
>120 PRINT "2 PROGRAMMA 2."
>130 PRINT "3 PROGRAMMA 3."
>140 PRINT "4 FINE."
>150 PRINT
>160 INPUT "SCELTA: ":C
>170 IF C=1 THEN RUN "DSK1.PR
 G1"
>180 IF C=2 THEN RUN "DSK1.PR
 G2"
>190 IF C=3 THEN RUN "DSK1.PR
 G3"
>200 IF C=4 THEN STOP
>210 GOTO 100
```

# SAVE

==============================================================================

**Formato**

SAVE *dispositivo.nome-programma* [,PROTECTED]
SAVE *dispositivo.nome-programma* [,MERGE]

**Descrizione**

Il comando SAVE permette di copiare in memoria dal *dispositivo* esterno il programma specificato in *nome programma*. Usando il comando OLD potrete poi richiamare quel programma in memoria. Il modo per registrare sul registratore a cassette e' spiegato nel manuale fornito con la consolle. Il modo per registrare sui dischi si trova sul manuale del *sistema di memoria a dischi*. Il SAVE cancella gli eventuali breakpoints inseriti nel programma.

**Opzioni**
Con i registratori a cassette e' possibile solo l'opzione PROTECTED.
Usando la parola chiave PROTECTED, si puo' facoltativamente specifcare che un programma puo' essere solo eseguito o richiamato in memoria mediante OLD. Il programma non puo' pero' essere listato, corretto ne' registrato. Questo non e' lo stesso modo di protezione possibile con il modulo Disk Manager. Nota: assicurarsi di conservare una copia non protetta dei programmi poiche' la protezione e' irreversibile. Se desiderate proteggere il programma affinche' non venga copiato, usare il sistema di protezione del modulo Disk Manager.
Potete facoltativamente specificare che il programma potra' essere fuso con un altro programma, usando la parola chiave MERGE. Soltanto i programmi salvati con l'opzione MERGE potranno essere fusi con altri programmi.

**Esempi**

SAVE DSK1.PRG1 salva il programma in memoria sul dischetto situato nel drive 1 sotto il nome PRG1.

```
>SAVE DSK1.PRG
```

SAVE DSK1.PRG1,PROTECTED salva il programma in memoria sul dischetto situato nel drive 1 con il nome PRG1. Il programma potra' essere chiamato in memoria ed eseguito, ma non potra' essere corretto, listato o risalvato.

```
>SAVE DSK1.PRG1,PROTECTED
```

SAVE DSK1.PRG1,MERGE il programma cosi' salvato potra' essere in seguito fuso con un programma in memoria.

```
>SAVE DSK1.PRG1,MERGE
```

---

# Sottoprogramma SAY

---

**Formato**

CALL SAY(*parola*[,*stringa-diretta*][,...])

**Descrizione**

Se e' collegato alla tastiera il sintetizzatore vocale (venduto separatamente), il sottoprogramma SAY da' la voce al computer. Esso ripete la *parola* o il valore specificato nella *stringa-diretta*. Per una completa descrizione di SAY vedere il manuale fornito insieme al modulo Speech Editor o al sintetizzatore vocale.
Il valore della *parola* e' un qualsiasi valore di stringa elencato nell'Appendice L. Se e' dato un valore letterale esso deve essere compreso tra virgolette. Il valore della *stringa-diretta* e' una valore restituito mediante il sottoprogramma SPGET. Il valore della *stringa-diretta* puo' essere cambiato aggiungendo i suffissi come descritto nell'Appendice M. Le parole e le *stringhe-dirette* devono essere alternate nel sottoprogramma CALL SAY. Se desiderate avere 2 *stringhe-dirette* o *parole* pronunciate consecutivamente potete inserirvi una virgola extra per indicare la posizione della voce omessa.

**Esempi**

CALL SAY("HELLO,HOW ARE YOU") fa pronunciare al computer "Hello how are you".

```
>100 CALL SAY("HELLO,HOW ARE
 YOU")
```

CALL SAY(,A$,,B$) fa in modo che il computer pronunci le parole indicate da A$ e B$, le quali devono essere riportate da SPGET.

```
>100 CALL SAY(,A$,,B$)
```

**Programma**

Il programma a destra dimostra l'uso di CALL SAY con una *parola* e 3 *stringhe-dirette*.

```
>100 CALL SPGET("HOW",X$)
>110 CALL SPGET("ARE",Y$)
>120 CALL SPGET("YOU",Z$)
>130 CALL SAY("HELLO",X$,,Y$,
 ,Z$)
```

## Sottoprogramma SCREEN

**Formato**

CALL SCREEN(*codice-colore*)

**Descrizione**

Il sottoprogramma SCREEN cambia i colori dello schermo con quelli specificati dal *codice-colore*. Tutte le parti dello schermo che non hanno caratteri su di esse, o che hanno caratteri o parti di caratteri con colore 1 (trasparente), sono mostrate nel colore specificato dal *codice-colore*. Il colore standard dello schermo con il TI Extended BASIC e' 8, ciano.

I codici dei colori sono:

| *Codice-colore* | *Codice-colore* |
|---|---|
| 1 Trasparente | 9 Rosso |
| 2 Nero | 10 Rosso chiaro |
| 3 Verde | 11 Giallo scuro |
| 4 Verde chiaro | 12 Giallo chiaro |
| 5 Blu scuro | 13 Verde scuro |
| 6 Blu chiaro | 14 Magenta |
| 7 Rosso scuro | 15 Grigio |
| 8 Ciano | 16 Bianco |

**Esempi**

CALL SCREEN((8) cambia lo schermo in ciano, che e' il colore standard.

```
>100 CALL SCREEN(9)
```

CALL SCREEN(2) cambia in nero il colore dello schermo.

```
>100 CALL SCREEN(2)
```

---

## SEG$

---

**Formato**

SEG$(*espressione-stringa,posizione,lunghezza*)

**Descrizione**

La funzione SEG$ restituisce la porzione di una stringa. La stringa restituita parte dalla *posizione* della *espressione-stringa* specificata fino alla *lunghezza* in caratteri specificata. Se la *posizione* e' oltre la fine dell'*espressione stringa*, viene restituita una stringa nulla (""). Se la *lunghezza* va oltre la lunghezza della stringa, sono restituiti solo i caratteri finali.

**Esempi**

X$=SEG$("FIRSTNAME LASTNAME",1,9) pone X$ uguale a "FIRSTNAME".

```
>100 X$=SEG$("FIRSTNAME LASTN
AME",1,9)
```

Y$=SEG$("FIRSTNAME LASTNAME",11,8) pone Y$ uguale "LASTNAME".

```
>100 Y$=SEG$("FIRSTNAME LASTN
AME",11,8)
```

Z$=SEG$("FIRSTNAME LASTNAME",10,1) pone Z$ uguale a " ".

```
>100 Z$=SEG$("FIRSTNAME LASTN
AME",10,1)
```

PRINT SEG$(A$,B,C) stampa la porzione della stringa A$ a partire dal carattere B per una estensione di C caratteri.

```
>100 PRINT SEG$(A$,B,C)
```

---

# SGN

===

**Formato**

SGN(*espressione-numerica*)

**Descrizione**

La funzione SGN restituisce 1 se l'*espressione-numerica* e' positiva, 0 se e' zero e -1 se e' negativa.

**Esempi**

IF SGN(X2)=1 THEN 300 ELSE 400
trasferisce il controllo alla linea 300 se X2 e' positivo, ed alla linea 400 se X2 e' uguale a zero o e' negativo.

```
>100 IF SGN(X2)=1 THEN 300 EL
 SE 400
```

ON SGN(X)+2 GOTO 200,300,400
trasferisce il controllo alla linea 200 se X e' negativo, alla linea 300 se X e' zero, e alla linea 400 se X e' positivo.

```
>100 ON SGN(X)+2 GOTO 200,300
 ,400
```

---

# SIN

---

**Formato**

SIN(*espressione-numerica*)

**Descrizione**

La funzione SIN da' il seno dell'*espressione-numerica* data in radianti. Se l'angolo e' espresso in gradi moltiplicare i gradi per PI/180 per avere l'equivalente angolo in radianti.

**Programma**

Il programma a destra calcola il seno di alcuni angoli.

```
>100 A=.5235987755982
>110 B=30
>120 C=45*PI/180
>130 PRINT SIN(A);SIN(B)
>140 PRINT SIN(B*PI/180)
>150 PRINT SIN(C)
>RUN
.5  -.9880316241
.5
.7071067812
```

---

# SIZE

---

**Formato**

SIZE

**Descrizione**

Il comando SIZE visualizza il numero dei bytes ancora disponibili in memoria. Se e' collegata l'espansione di memoria il numero dei bytes liberi e' dato dalla quantita' libera dello stack e da quella dell'area riservata ai programmi. Un byte e' lo spazio di memoria richiesto per qualsiasi cosa come un carattere, una cifra o una parola riservata del TI Extended BASIC.

Se l'espansione di memoria non e' collegata, lo spazio disponibile e' dato dalla quantita' di spazio lasciata libera da quella occupata dal programma, dallo schermo, dalla tavola degli sprites, dalla tavola dei colori e dai valori di stringa.

Se l'espansione di memoria e' collegata, lo spazio disponibile nello stack e' la quantita' di spazio rimasta libera dopo l'inserimento di valori di stringa e di informazioni sulle variabili. Lo spazio per il programma e' la quantita' di spazio rimasta libera dopo l'inserimento del programma e dei valori delle variabili numeriche.

**Esempi**

SIZE da' la memoria libera.

```
>SIZE
    13928 BYTES FREE
```

SIZE da' la memoria libera. Se e' collegata l'espansione di memoria questa e' suddivisa nelle 2 parti dello stack e dell'area programma.

```
>SIZE
    13928 BYTES OF STACK FREE
    24511 BYTES OF PROGRAM
          SPACE FREE
```

---

## Sottoprogramma SOUND

---

**Formato**

CALL SOUND(*durata,frequenza1,volume1*[,...,*frequenza4,volume4*])

**Descrizione**

Il sottoprogramma SOUND dice al computer di produrre suoni o rumori. I valori dati riguardano il controllo di tre aspetti del suono: *durata, frequenza* e *volume*.

| *Valore* | *Ampiezza* | *Descrizione* |
|---|---|---|
| Durata | da 1 a 4250<br>-1 -4250 | La lunghezza del suono in millesimi di secondo. |
| Frequenza | (Tono) da 110 a 44733<br>(Rumore) -1 -8 | Quale suono e' emesso. |
| Volume | da 0 a 30 | Volume del suono. |

La *durata* varia da .001 a 4.250 secondi, sebbene essa possa variare fino ad 1/60.mo di secondo. Il computer continua ad eseguire le istruzioni del programma mentre viene riprodotto il suono. Quando chiamate il sottoprogramma SOUND, il computer attende finche' il precedente suono sia stato completato prima di eseguire una nuova CALL SOUND. Tuttavia se e' specificata una *durata* negativa, il precedente suono si ferma e viene eseguito subito dopo quello nuovo.
La *frequenza* specifica la frequenza della nota che deve essere riprodotta con un valore compreso fra 110 e 44733. (NOTA: questa gamma va oltre il limite dell'udito umano. C'e' una differente capacita' personale di ascoltare le note alte, ma generalmente la piu' alta ha un valore di 10000). La reale frequenza prodotta dal computer puo' variare fino al dieci per cento da quella reale. L'Appendice D contiene l'elenco delle frequenze delle note piu' comuni.
Un valore da -1 a -8 specifica uno degli 8 diversi tipi di rumore.

| *Frequenza* | *Descrizione* |
|---|---|
| -1 | rumore periodico del 1.o tipo |
| -2 | rumore periodico del 2.o tipo |
| -3 | rumore periodico del 3.o tipo |
| -4 | rumore periodico che varia con la frequenza del 3.o tono specificato |
| -5 | rumore bianco del 1.o tipo |
| -6 | rumore bianco del 2.o tipo |
| -7 | rumore bianco del 3.o tipo |
| -8 | rumore bianco che varia con la frequenza del 3.o tono specificato |

Si possono riprodurre contemporaneamente fino ad un massimo di tre toni ed un rumore.
Il *volume* specifica l'intensita' della nota o del rumore. Zero e' il piu' alto e 30 il piu' basso.

**Esempi**

CALL SOUND(1000,110,0) riproduce il LA due ottave sotto il DO centrale per una durata di un secondo e con il massimo volume.

```
>100 CALL SOUND(1000,110,0)
```

CALL SOUND(500,110,0,131,0,196,3) riproduce il LA due ottave sotto il DO centrale, il DO un'ottava sotto il DO centrale ed il SOL un'ottava sotto il DO centrale per la durata di mezzo secondo con il massimo volume.

```
>100 CALL SOUND(500,110,0,131
 ,0,196,3)
```

CALL SOUND(4250,-8,0) riproduce un rumore per la durata di 4,250 secondi.

```
>100 CALL SOUND(4250,-8,0)
```

CALL SOUND(DUR,TONO,VOL) riproduce il tono indicato in TONO per la durata indicata in DUR e con il volume indicato in VOL.

```
>100 CALL SOUND(DUR,TONO,VOL)
```

**Programma**

Il programma sulla destra riproduce le 13 note della prima ottava che sono disponibili nel computer.

```
>100 X=2^(1/12)
>110 FOR A=1 TO 13
>120 CALL SOUND(100,110*X^A,0
 )
>130 NEXT A
```

---

# Sottoprogramma SPGET

---

**Formato**

CALL SPGET(*parola*,*variabile-stringa*)

**Descrizione**

Il sottoprogramma SPGET riporta nella *variabile-stringa* la frase che corrisponde alla *parola*. Per una completa descrizione di SPGET vedere il manuale fornito con il modulo di comando Speech Editor o con il sintetizzatore vocale (venduti separatamente).

Il valore della *parola* corrisponde a ciascun valore di stringa fra quelli elencati nell'Appendice L. Se viene dato un valore in lettere esso deve essere racchiuso tra apici. Il valore della *variabile-stringa* e' usato con SAY, e puo' essere alterato per raggiungere dei suffissi come descritto nell'Appendice M.

**Programma**

Il programma a destra illustra l'uso di CALL SPGET.

```
>100 CALL SPGET("HOW",X$)
>110 CALL SPGET("ARE",Y$)
>120 CALL SPGET("YOU",Z$)
>130 CALL SAY("HELLO",X$,,Y$,
 ,Z$)
```

---

# Sottoprogramma SPRITE

---

**Formato**

CALL SPRITE(#*numero-sprite*,*codice-carattere*,*colore-sprite*,*pixel-riga*,*pixel-colonna* [,*vel-riga*,*vel-colonna*][,...])

**Descrizione**

Il sottoprogramma SPRITE crea gli sprites. Gli sprites sono disegni che hanno un proprio colore ed occupano una qualsiasi locazione dello schermo. Essi possono essere posti in movimento ed in qualsiasi direzione e con velocita' variabili, e continuano il loro movimento finche' esso non viene cambiato dal programma o al termine di questo. Essi si muovono con piu' uniformita' del solito carattere che salta da una posizione dello schermo all'altra.
Il *numero-sprite* e' un'epressione numerica compresa tra 1 e 28. Se il valore e' quello di uno sprite gia' definito il vecchio sprite viene cancellato e sostituito da quello nuovo. Se il vecchio sprite ha una velocita' di *-riga* o di *-colonna* e quello nuovo non le specifica, quest'ultimo assumera' le vecchie velocita'.
Gli sprites passano sopra agli altri caratteri presenti sullo schermo senza cancellarli. Quando due o piu' sprites sono coincidenti lo sprite con il numero piu' basso copre gli altri. Mentre cinque o piu' sprites si trovano sulla stessa riga dello schermo, quello o quelli con il piu' alto *numero-sprite* spariscono.
Il *codice-carattere* puo' essere un numero intero compreso fra 32 e 143. Vedere il sottoprogramma CHAR per informazioni sulla definizione dei caratteri. Il *codice-carattere* puo' essere cambiato mediante il sottoprogramma PATTERN. Lo sprite e' definito dal carattere dato e, nel caso di sprite con dimensione doppia, dai successivi tre caratteri. Vedere il sottoprogramma MAGNIFY per maggiori informazioni.
Il *colore-sprite* puo' essere una qualsiasi espressione numerica tra 1 e 16. Essa determina il colore di primo piano dello sprite (foreground). Il colore di sfondo (background) di uno sprite e' sempre 1, trasparente. Vedere i sottoprogrammi COLOR e SCREEN per maggiori informazioni.
*Pixel-riga* e *pixel-colonna* sono numerati consecutivamente a partire da 1 nell'angolo superiore sinistro dello schermo. Il *pixel-riga* puo' variare da 1 a 192 e il *pixel-colonna* da 1 a 256 (in realta' i *pixels-riga* arrivano fino a 256, ma le posizioni da 193 a 256 sono al di fuori della portata del margine inferiore dello schermo). La posizione dello sprite e' l'angolo superiore sinistro del carattere o dei caratteri che lo definiscono.

Informazioni sulla posizione di uno sprite possono essere trovate mediante l'utilizzo dei sottoprogrammi POSITION, COINC e DISTANCE. La locazione di uno sprite puo' essere cambiata usando il sottoprogramma LOCATE. COLOR cambia il colore di uno sprite. Gli sprites possono essere eliminati con il sottoprogramma DELSPRITE. Quando avviene un breakpoint o il programma termina, gli sprites cessano di esistere e non ricompariranno con CONTINUE.

**Opzioni**
*Velocita'-riga* e *velocita'-colonna* possono essere facoltativamente specificate al momento della creazione dello sprite. Se sono entrambe uguali a zero, lo sprite risulta fermo. Una *velocita'-riga* positiva sposta lo sprite verso il basso ed una negativa lo muove verso l'alto. Una *velocita'-colonna* positiva sposta lo sprite verso destra ed una negativa lo sposta verso sinistra. Se intrambe non sono uguali a zero lo sprite si muove diagonalmente in una direzione che dipende dai valori assegnati.
Le velocita' di *riga* e di *colonna* possono variare tra -128 e 127. Con un valore vicino allo 0 la velocita' risulta molto lenta. Man mano che essa si allontana dallo 0, aumenta progressivamente la velocita'. Quando uno sprite arriva ad un'estremita' dello schermo, esso sparisce e ricompare nella posizone corrispondente sull'altro lato. La velocita' di uno sprite puo' essere cambiata usando il sottoprogramma MOTION.

**Programmi**
I tre programmi seguenti mostrano alcuni possibili usi degli sprites. Il terzo usa tutti i sottoprogrammi che hanno relazione con gli sprites, eccettuato COLOR e DISTANCE.

```
>100 CALL CLEAR
>110 CALL CHAR(96,"FFFFFFFFFF
 FFFFFF")
>120 CALL CHAR(98,"183C7EFFFF
 7E3C18")
>130 CALL CHAR(100,"F00FF00FF
 00FF00F")
```

La linea 140 crea uno sprite di colore blu scuro al centro dello schermo ed uno rosso scuro nell'angolo superiore sinistro. La linea 150 crea uno sprite bianco vicino all'angolo superiore destro dello schermo che comincia a muoversi lentamente di 45 gradi in basso a destra. Lo sprite e' un punto esclamativo.

```
>140 CALL SPRITE(#1,96,5,92,1
 24,#2,100,7,1,1)
>150 CALL SPRITE(#28,33,16,12
 ,48,1,1)
```

La linea 160 crea uno sprite nell'angolo superiore sinistro dello schermo che comincia a muoversi velocemente di 45 gradi in alto a destra.

```
>160 CALL SPRITE(#15,98,14,1,
 1,127,-128)
>170 GOTO 170
```

(Premere **FCTN 4** per fermare il programma).

Il programma sulla destra fa uso degli sprites creando degli effetti particolari. La linea 110 definisce il carattere 96. La linea 150 definisce gli sprites, che in tutto sono 28. Il *numero-sprite* e' il valore che ha A in quel momento. Il *codice-carattere* e' il 96. Il *colore-sprite* e' dato dall'espressione INT(A/3)+3 . I punti di partenza orizzontali e verticali sono i pixels 92 e 124 corrispondenti al centro dello schermo. Le velocita' di riga e di colonna sono scelte casualmente usando il valore di A*INT(RND*4.5) -2.25+A/2*SGN(RND-.5) . La linea 170 fa in modo che sia ripetuta la sequenza.

```
>100 CALL CLEAR
>110 CALL CHAR(96,"0008081C7F
 1C0808")
>120 RANDOMIZE
>130 CALL SCREEN(2)
>140 FOR A=1 TO 28
>150 CALL SPRITE(#A,96,INT(A/
 3)+3,92,124,A*INT(RND*4.5)-2
 .25+A/2*SGN(RND-.5),A*INT(RN
 D*4.5)-2.25+A/2*SGN(RND-.5))
>160 NEXT A
>170 GOTO 140
```

(Premere **FCTN 4** per fermare il programma).

Il seguente programma fa uso di tutti i sottoprogrammi relativi agli sprites eccetto che COLOR e DISTANCE. Essi sono CHAR, COINC, DELSPRITE, LOCATE, MAGNIFY, MOTION, PATTERN, POSITION e SPRITE. Il programma crea 2 sprites ingranditi del doppio con la forma di una persona che cammina lungo un piano. C'e' una barriera che uno di loro attraversa e che l'altro salta. Quello che salta va un po' piu' veloce dopo ciascun salto, finche' non raggiunge l'altro. Quando avviene cio' essi vengono rimpiccoliti della meta' e continuano a camminare. Quando si incontrano per la seconda volta, quello che stava andando piu' veloce sparisce, e l'altro continua a camminare.

Le linee 110, 120, 140, 150, 250 e 260 definiscono gli sprite.

```
>100 CALL CLEAR
>110 S1$="0103030103030303030
 303030303030380C0C080C0C0C0C
 0C0C0C0C0C0C0C0E0"
>120 S2$="01030301030700F181B0
 303030600C0C0EB0C0C080C0E0F0D
 ECCC0C0C0603030 3B"
```

La linea 130 pone a 0 il contatore degli incontri.

```
>130 COUNT=0
>140 CALL CHAR(96,S1$)
>150 CALL CHAR(100,S2$)
>160 CALL SCREEN(14)
```

Le linee dalla 170 alla 200 costruiscono il piano.

```
>170 CALL COLOR(14,13,13)
>180 FOR A=19 TO 24
>190 CALL HCHAR(A,1,136,32)
>200 NEXT A
```

Le linee dalla 210 alla 240 costruiscono la barriera.

```
>210 CALL COLOR(13,15,15)
>220 CALL VCHAR(14,22,128,6)
>230 CALL VCHAR(14,23,128,6)
>240 CALL VCHAR(14,24,128,6)
>250 CALL SPRITE(#1,96,5,113
 ,129,#2,96,7,113,9)
>260 CALL MAGNIFY(4)
```

La linea 270 pone la velocita' di partenza dello sprite che poi accelerera'.

```
>270 XDIR=4
>280 PAT=2
```

La linea 290 mette in movimento gli sprites.

```
>290 CALL MOTION(#1,0,XDIR,#2
 ,0,4)
```

La linea 300 crea l'illusione del movimento.

```
>300 CALL PATTERN(#1,98+PAT,#
 2,98-PAT)
>310 PAT=-PAT
```

La linea 320 verifica l'incontro tra i due sprites.

```
>320 CALL COINC(ALL,CO)
```

La linea 330 trasferisce il controllo se gli sprites si sono incontrati. Le linee 340 e 350 verificano che lo sprite abbia raggiunto la barriera, e trasferiscono il controllo in caso positivo.

```
>330 IF CO<>0 THEN 370
>340 CALL POSITION(#1,YPOS1,X
 POS1)
>350 IF XPOS1>136 AND XPOS1<1
 92 THEN 470
```

La linea 360 rimanda indietro per continuare il movimento.

Le linee dalla 370 alla 460 guidano gli sprites ad incontrarsi.

Le linee 380 e 390 li fermano.

```
>360 GOTO 300
>370 REM COINCIDENZE
>380 CALL MOTION(#1,0,0,#2,0,
 0)
>390 CALL PATTERN(#1,96,#2,96
 )
```

La linea 400 verifica il primo incontro. La linea 410 incrementa il contatore degli incontri. La linea 420 calcola la loro posizione.

```
>400 IF COUNT>0 THEN 540
>410 COUNT=COUNT+1
>420 CALL POSITION(#1,YPOS1,X
 POS1,#2,YPOS2,XPOS2)
```

La linea 430 li impiccolisce.

La linea 440 li mette sul piano e muove quello veloce leggermente in avanti.

```
>430 CALL MAGNIFY(3)
>440 CALL LOCATE(#1,YPOS1+16,
 XPOS1+8,#2,YPOS2+16,XPOS2)
```

La linea 450 fa riprendere il loro movimento.

```
>450 CALL MOTION(#1,0,XDIR,#2
 ,0,4)
>460 GOTO 340
```

Le linee dalla 470 alla 530 portano lo sprite piu' veloce a saltare la barriera. La linea 480 lo ferma.

```
>470 REM #1 COLPISCE MURO
>480 CALL MOTION(#1,0,0)
>490 CALL POSITION(#1,YPOS1,X
 POS1)
```

La linea 500 lo pone nella nuova posizione oltre la barriera.

```
>500 CALL LOCATE(#1,YPOS1,193
 )
```

Le linee 510 e 520 fanno riprendere il movimento, un po' piu' veloce.

```
>510 XDIR=XDIR+1
>520 CALL MOTION(#1,0,XDIR)
>530 GOTO 300
```

Le linee dalla 540 alla 640 guidano il secondo incontro.

```
>540 REM SECONDA COINCIDENZA
>550 FOR DELAY=1 TO 500 :: NE
 XT DELAY
```

La linea 560 fa partire lo sprite lento, mentre la 570 cancella quello veloce. Le linee dalla 580 alla 630 fanno fare allo sprite lento 20 passi.

```
>560 CALL MOTION(#2,0,4)
>570 CALL DELSPRITE(#1)
>580 FOR STEP1=1 TO 20
>590 CALL PATTERN(#2,100)
>600 FOR DELAY=1 TO 20 :: NEX
 T DELAY
>610 CALL PATTERN(#2,96)
>620 FOR DELAY=1 TO 20 :: NEX
 T DELAY
>630 NEXT STEP1
>640 CALL CLEAR
```

168 - 220

---

## SQR

---

**Formato**

SQR(*espressione-numerica*)

**Descrizione**

La funzione SQR restituisce la radice quadrata positiva del-l'*espressione-numerica*. SQR(X) e' equivalente ad XA(1/2). L'*espressione-numerica* non puo' essere un numero negativo.

**Esempi**

PRINT SQR(4) stampa 2.

```
>PRINT SQR(4)
```

X=SQR(2.57E5) pone X uguale alla radice quadrata di 257000, che e' 506.9516742 .

```
>X=SQR(2.57E5)
```

---

## STOP

---

**Formato**

STOP

**Descrizione**

L'istruzione STOP ferma l'esecuzione del programma. Essa puo' essere usata al posto di END con la differenza che non puo' essere posto alla fine dei sottoprogrammi.

**Programma**

Il programma sulla destra illustra l'uso dell'istruzione STOP. Il programma somma i numeri da 1 a 100.

```
>100 CALL CLEAR
>110 TOT=0
>120 NUMB=1
>130 TOT=TOT+NUMB
>140 NUMB=NUMB+1
>150 IF NUMB>100 THEN PRINT T
 OT::STOP
```

# STR$

**Formato**

STR$(*espressione-numerica*)

**Descrizione**

La funzione STR$ trasforma in stringa l'equivalente dell'*espressione-numerica*. Questo permette alle funzioni, alle istruzioni ed ai comandi che riguardano le stringhe, di essere trattati nella rappresentazione in caratteri dell'*espressione-numerica*. La funzione STR$ e' l'inverso della funzione VAL.

**Esempi**

NUM$=STR$(78.6) pone NUM$ uguale a "78.6" .

```
>100 NUM$=STR$(78.6)
```

LL$=STR$(3E15) pone LL$ uguale a "3E15" .

```
>100 LL$=STR$(3E15)
```

I$=STR$(A*4) pone I$ uguale alla stringa con qualsiasi valore si ottenga moltiplicando A per 4 . Per esempio, se A e' uguale a -8, I$ sara' uguale a "-32" .

```
>100 I$=STR$(A*4)
```

---

## SUB

---

**Formato**

SUB *nome-sottoprogramma* [(*lista-parametri*)]

**Descrizione**

L'istruzione SUB e' la prima istruzione di un sottoprogramma. I sottoprogrammi sono usati quando si desidera separare un gruppo d'istruzioni dal programma principale. Si possono usare i sottoprogrammi per eseguire una stessa operazione piu' volte in un programma, in diversi programmi o per usare variabili che sono specifiche del sottoprogramma. L'istruzione SUB non puo' trovarsi in un'istruzione IF-THEN-ELSE.
I sottoprogrammi sono chiamati con CALL *nome-sottoprogramma* [(*lista-parametri*)] . I sottoprogrammi devono terminare con SUBEND e lasciati o con la stessa SUBEND o con l'istruzione SUBEXIT. Il controllo viene trasferito all'istruzione successiva a quella che ha chiamato il sottoprogramma. Non si deve mai trasferire il controllo all'esterno del sottoprogramma con altre istruzioni diverse da SUBEND e SUBEXIT. Cio' comporta il passaggio del controllo con ON ERROR.
Quando un programma contiene un sottoprogramma, quest'ultimo deve trovarsi in coda al programma principale. La struttura di un programma deve essere come segue:

| | |
|---|---|
| Inizio programma principale | |
| - | |
| - | |
| - | |
| Chiamate sottoprogrammi | |
| - | |
| - | |
| Fine del programma principale | Il programma si fermera' qui senza un'istruzione STOP o END. |
| Inizio primo sottoprogramma | I sottoprogrammi sono opzionali. |
| - | |
| - | |
| - | |
| Fine del primo sottoprogramma | Non deve apparire niente tra i sottoprogrammi eccetto istruzioni REM e END. |
| Inizio secondo sottoprogramma | |
| - | |
| - | |
| Fine secondo sottoprogramma | Dopo i sottoprogrammi possono comparire solo commenti e END. |
| Fine del programma | |

---

# STR$

===

**Formato**

STR$(*espressione-numerica*)

**Descrizione**

La funzione STR$ trasforma in stringa l'equivalente dell'*espressione-numerica*. Questo permette alle funzioni, alle istruzioni ed ai comandi che riguardano le stringhe, di essere trattati nella rappresentazione in caratteri dell'*espressione-numerica*. La funzione STR$ e' l'inverso della funzione VAL.

**Esempi**

| | |
|---|---|
| NUM$=STR$(78.6) pone NUM$ uguale a "78.6" . | >100 NUM$=STR$(78.6) |
| LL$=STR$(3E15) pone LL$ uguale a "3E15" . | >100 LL$=STR$(3E15) |
| I$=STR$(A*4) pone I$ uguale alla stringa con qualsiasi valore si ottenga moltiplicando A per 4 . Per esempio, se A e' uguale a -8, I$ sara' uguale a "-32" . | >100 I$=STR$(A*4) |

---

# SUB

===

**Formato**

SUB *nome-sottoprogramma* [(*lista-parametri*)]

**Descrizione**

L'istruzione SUB e' la prima istruzione di un sottoprogramma. I sottoprogrammi sono usati quando si desidera separare un gruppo d'istruzioni dal programma principale. Si possono usare i sottoprogrammi per eseguire una stessa operazione piu' volte in un programma, in diversi programmi o per usare variabili che sono specifiche del sottoprogramma. L'istruzione SUB non puo' trovarsi in un'istruzione IF-THEN-ELSE.
I sottoprogrammi sono chiamati con CALL *nome-sottoprogramma* [(*lista-parametri*)] . I sottoprogrammi devono terminare con SUBEND e lasciati o con la stessa SUBEND o con l'istruzione SUBEXIT. Il controllo viene trasferito all'istruzione successiva a quella che ha chiamato il sottoprogramma. Non si deve mai trasferire il controllo all'esterno del sottoprogramma con altre istruzioni diverse da SUBEND e SUBEXIT. Cio' comporta il passaggio del controllo con ON ERROR.
Quando un programma contiene un sottoprogramma, quest'ultimo deve trovarsi in coda al programma principale. La struttura di un programma deve essere come segue:

| | |
|---|---|
| Inizio programma principale<br>-<br>-<br>-<br>Chiamate sottoprogrammi<br>-<br>- | |
| Fine del programma principale | Il programma si fermera' qui senza un'istruzione STOP o END. |
| Inizio primo sottoprogramma<br>-<br>-<br>- | I sottoprogrammi sono opzionali. |
| Fine del primo sottoprogramma | Non deve apparire niente tra i sottoprogrammi eccetto istruzioni REM e END. |
| Inizio secondo sottoprogramma<br>-<br>- | |
| Fine secondo sottoprogramma | Dopo i sottoprogrammi possono comparire solo commenti e END. |
| Fine del programma | |

**Opzioni**

Tutte le variabili usate in un sottoprogramma, escluse quelle incluse nella *lista-parametri*, sono limitate a quel sottoprogramma, cosicche' e' possibile usare gli stessi nomi di variabili usati nel programma principale o in altri sottoprogrammi ed alterarne i valori senza alcun effetto sulle altre variabili. In modo analogo, i valori delle variabili nel programma principale o negli altri sottoprogrammi non hanno effetto sui valori delle variabili del sottoprogramma (tuttavia, le istruzioni DATA sono utilizzabili dai sottoprogrammi).

Lo scambio di valori tra il programma principale ed il sottoprogramma avviene con l'opzionale *lista-parametri*. I parametri non hanno bisogno di avere gli stessi nomi dell'istruzione chiamante, ma devono essere dati dello stesso tipo (numerici o alfanumerici), e nello stesso ordine delle voci nella CALL. Se in un sottoprogramma vengono modificati i valori di variabili semplici trasferite dal programma principale, essi vengono modificati anche all'interno del programma principale. Cosi', il valore dell'elemento di una matrice, come A(1), specificato nella lista parametri dell'istruzione chiamante, viene modificato anche nel programma principale quando il controllo ritorna all'istruzione successiva a quella chiamante.

Il valore derivato da un'espressione in un'istruzione chiamante, che viene modificato all'interno del sottoprogramma, non cambia all'interno del programma principale. Le intere matrici sono trasferite mediante riferimento, cosi' gli elementi che cambiano all'interno del sottoprogramma cambiano i loro valori anche all'interno della matrice nel programma principale. Le matrici sono indicate facendo seguire il nome del parametro tra parentesi. Se la matrice e' a piu' di una dimensione deve essere inserita tra le parentesi una virgola per ciascun dimensionamento addizionale.

Se desiderate, potete trasferire valori soltanto per variabili semplici racchiudendole tra parentesi. Allora il valore puo' essere usato nel sottoprogramma, ma esso non viene cambiato al ritorno nel programma principale. Per esempio, CALL SPRG1((A)) trasferisce il valore di A ad un sottoprogramma che inizia con SUB SPRG1(X), e permette a quel valore di essere usato in X, senza cambiare il valore di A nel programma principale se il sottoprogramma cambia il valore di X.

Se un sottoprogramma e' chiamato piu' di una volta, qualsiasi variabile locale usata nel sottoprogramma conserva quei valori da una chiamata all'altra.

**Esempi**

SUB MENU indica l'inizio di un sottoprogramma. Non sono trasferiti ne' riportati parametri.

```
>100 SUB MENU
```

SUB MENU(COUNT,CHOICE) indica l'inizio di un sottoprogramma. Le variabili COUNT e CHOICE possono essere usate e/o avere i loro valori cambiati nel sottoprogramma e riportati alle variabili nella stessa posizione dell'istruzione chiamante.

```
>100 SUB MENU(COUNT,CHOICE)
```

SUB PAYCHECK(DATE,Q,SSN,PAYRATE, TABLE(,)) indica l'inizio di un sottoprogramma. Le variabili DATE, Q, SSN, PAYRATE e la matrice a due dimensioni TABLE possono essere usate e/o avere i loro valori cambiati nel sottoprogramma e riportati nelle variabili nella stessa posizione dell'istruzione chiamante.

```
>100 SUB PAYCHECK(DATE,Q,SSN,
 PAYRATE,TABLE(,))
```

**Programma**

Il programma a destra illustra l'uso di SUB. Il sottoprogramma MENU e' stato precedentemente salvato con l'opzione MERGE. Esso stampa un menu che richiede una scelta. Il programma principale chiede al sottoprogramma quante scelte ci sono e quali. Quindi usa la scelta fatta nel sottoprogramma per determinare quale programma deve essere lanciato.

```
>100 CALL MENU(5,R)
>110 ON R GOTO 120,130,140,15
 0,160
>120 RUN "DSK1.PAYABLES"
>130 RUN "DSK1.RECEIVE"
>140 RUN "DSK1.PAYROLL"
>150 RUN "DSK1.INVENTORY"
>160 RUN "DSK1.LEDGER"
>170 DATA ACCOUNTS PAYABLE,AC
 COUNTS RECEIVABLE,PAYROLL,IN
 VENTORY,GENERAL LEDGER
```

Inizio del sottoprogramma MENU.

Notare che questa R non e' la stessa R usata nelle linee 100 e 110 nel programma principale.

```
>10000 SUB MENU(COUNT,CHOICE)
>10010 CALL CLEAR
>10020 IF COUNT>22 THEN PRINT
  "TOO MANY ITEMS" :: CHOICE=
 0 :: SUBEXIT
>10030 RESTORE
>10040 FOR R=1 TO COUNT
>10050 READ TEMP$
>10060 TEMP$=SEG$(TEMP$,1,25)
>10070 DISPLAY AT(R,1):R;TEMP
 $
>10080 NEXT R
>10090 DISPLAY AT(R+1,1):"YOU
 R CHOICE: 1"
>10100 ACCEPT AT(R+1,14)BEEP
 VALIDATE(DIGIT)SIZE(-2):CHOI
 CE
>10110 IF CHOICE<1 OR CHOICE>
 COUNT THEN 10100
>10120 SUBEND
```

---

## SUBEND

---

**Formato**

SUBEND

**Descrizione**

L'istruzione SUBEND indica la fine di un sottoprogramma. Quando viene eseguita, il controllo e' trasferito all'istruzione successiva a quella che ha chiamato il sottoprogramma. L'istruzione SUBEND deve essere sempre l'ultima istruzione di un sottoprogramma. L'istruzione SUBEND non puo' comparire in un'istruzione IF-THEN-ELSE. Le uniche istruzioni che possono immediatamente seguire l'istruzione SUBEND sono REM, END o l'istruzione SUB per un successivo sottoprogramma.

---

## SUBEXIT

---

**Formato**

SUBEXIT

**Descrizione**

L'istruzione SUBEXIT permette di lasciare un sottoprogramma prima della sua fine (indicata con SUBEND). Quando essa e' eseguita il controllo viene passato all'istruzione successiva a quella che ha chiamato il sottoprogramma. L'istruzione SUBEXIT non ha bisogno di essere sempre presente in un sottoprogramma.

# TAB

---

**Formato**

TAB(*espressione-numerica*)

**Descrizione**

La funzione TAB specifica la posizione di partenza della successiva *voce-stampa* in una istruzione PRINT, PRINT...USING, DISPLAY o DISPLAY...USING . Se l'*espressione-numerica* e' maggiore della lunghezza di un record per il dispositivo al quale si riferisce la scrittura (per esempio: 28 per lo schermo, 32 per la stampante termica, il valore specificato per un file su dischetto o cassetta), allora essa viene ripetutamente ridotta dalla lunghezza del record finche' non rientra tra 1 e la lunghezza consentita.
Se il numero dei caratteri gia' stampati sull'attuale record e' minore o uguale dell'*espressione-numerica*, la successiva voce di stampa e' stampata a partire dalla posizione indicata dall'*espressione-numerica*. Se il numero dei caratteri gia' stampati sull'attuale record e' maggiore o uguale della posizione indicata dall'*espressione-numerica*, la successiva voce di stampa e' stampata sul record successivo a cominciare dalla posizione indicata nell'*espressione-numerica*. La funzione TAB e' trattata come una *voce-stampa*, quindi essa deve essere preceduta o seguita da un separatore di stampa (virgola, punto e virgola o due punti). Il separatore di stampa prima di TAB e' calcolato prima della funzione TAB. Normalmente prima e dopo TAB viene usato il punto e virgola.

**Esempi**

PRINT TAB(12);35 stampa il numero 35 alla dodicesima posizione.

```
>100 PRINT TAB(12);35
```

PRINT 356;TAB(18);"NOME" stampa 356 all'inizio della linea e NOME alla diciottesima posizione della linea.

```
>100 PRINT 356;TAB(18);"NOME"
```

PRINT "ABCDEFGHIJKLM";TAB(5); "NOP" stampa ABCDEFGHIJKLM all'inizio della linea e NOP alla quinta posizione della linea successiva.

```
>100 PRINT "ABCDEFGHIJKLM";TA
 B(5);"NOP"
```

DISPLAY AT(12,1):"NOME";TAB(15); "INDIRIZZO" visualizza NOME all'inizio della dodicesima riga dello schermo e INDIRIZZO alla quindicesima posizione sulla dodicesima riga dello schermo.

```
>100 DISPLAY AT(12,1):"NOME";
 TAB(15);"INDIRIZZO"
```

## TAN

**Formato**

TAN(*espressione-numerica*)

**Descrizione**

La funzione TAN calcola la tangente trigonometrica dell'*espressione-numerica* espressa in radianti. Se l'angolo e' in gradi, moltiplicare il numero dei gradi per PI/180 per ottenere l'equivalente angolo in radianti.

**Programma**

Il programma a destra calcola la tangente di alcuni angoli.

```
>100 A=.7853981633973
>110 B=26.565051177
>120 C=45*PI/180
>130 PRINT TAN(A);TAN(B)
>140 PRINT TAN(B*PI/180)
>150 PRINT TAN(C)
>RUN
 1.   7.17470553
.5
 1
```

## TRACE

**Formato**

TRACE

**Descrizione**

Il comando TRACE fa in modo che durante l'esecuzione di un programma siano visualizzati i numeri delle linee in procinto di essere eseguite. Questo vi permette di seguire il corso del programma per la messa a punto e per la ricerca degli errori. Il comando TRACE puo' essere usato come istruzione. L'effetto del comando TRACE e' eliminato quando sono eseguiti i comandi NEW o UNTRACE. Naturalmente NEW elimina anche il programma dalla memoria.

**Esempio**

TRACE fa in modo che siano visualizzate le linee del programma sullo schermo.

```
>TRACE
>100 TRACE
```

---

## UNBREAK

---

**Formato**

UNBREAK [*elenco-linee*]

**Descrizione**

Il comando UNBREAK rimuove tutti i breakpoints. Esso puo' essere facoltativamente assegnato solo per quelle linee specificate nel-l'*elenco-linee*. UNBREAK puo' essere usato come istruzione.

**Esempi**

| | |
|---|---|
| UNBREAK rimuove tutti i breakpoints. | >UNBREAK<br>>420 UNBREAK |
| UNBREAK 100,130 elimina i punti di arresto dalle linee 100 e 130. | >UNBREAK 100,130<br>>320 UNBREAK 100,130 |

---

## UNTRACE

---

**Formato**

UNTRACE

**Descrizione**

Il comando UNTRACE elimina l'effetto del comando TRACE. UNTRACE puo' essere usato come istruzione.

**Esempio**

| | |
|---|---|
| UNTRACE elimina l'effetto di TRACE | >UNTRACE<br>>420 UNTRACE |

---

## VAL

---

**Formato**

VAL(*espressione-stringa*)

**Descrizione**

La funzione VAL trasforma in valore numerico il contenuto di una variabile di stringa specificata in *espressione-stringa*. Questo permette alle funzioni, alle istruzioni ed ai comandi che riguardano i numeri di essere usati con l'*espressione-stringa*. Questa naturalmente deve contenere delle cifre. La funzione VAL e' l'inverso della funzione STR$.

**Esempi**

| | |
|---|---|
| NUMERO=VAL("78.6") pone NUMERO uguale a 78.6 . | >100 NUMERO=VAL("78.6") |
| LL=VAL(A$) pone LL uguale a 3E15 se A$ e' uguale a "3E15" . | >100 LL=VAL(A$) |

---

## Sottoprogramma VCHAR

---

**Formato**

CALL VCHAR(*riga*,*colonna*,*codice-carattere* [,*ripetizioni*])

**Descrizione**

Il sottoprogramma VCHAR posiziona un carattere in un qualsiasi punto dello schermo e lo ripete opzionalmente in verticale. Il carattere con il valore ASCII e' posto nella posizione specificata in *riga* e *colonna* ed e' ripetuto verticalmente per il numero delle *ripetizioni*.

Il valore di 1 per *riga* indica il bordo superiore dello schermo. Il valore di 24 e' il bordo inferiore. Il valore di 1 per *colonna* indica il lato sinistro dello schermo, ed il 32 quello destro. Lo schermo puo' essere immaginato come la griglia disegnata nella pagina accanto.

**Esempi**

| | |
|---|---|
| CALL VCHAR(12,16,33) posiziona il carattere 33 (un punto esclamativo) a riga 12 e colonna 16. | >100 CALL VCHAR(12,16,33) |
| CALL VCHAR(1,1,ASC("!"),768) pone un punto esclamativo a riga 1 e colonna 1 e lo ripete per 768 volte fino a riempire lo schermo. | >100 CALL VCHAR(1,1,ASC("!"),<br>768 |
| CALL VCHAR(R,C,K,T) pone il caratere con il codice ASCII di K a riga R e colonna C, e lo ripete T volte. | >100 CALL VCHAR(R,C,K,T) |

---

# Sottoprogramma VERSION

---

**Formato**

CALL VERSION(*variabile-numerica*)

**Descrizione**

Il sottoprogramma VERSION restituisce un valore che indica la versione del BASIC che si sta usando.
Con il TI Extended BASIC viene restituito un valore di 100.

**Esempio**

CALL VERSION(V) pone V uguale a 100.

```
>100 CALL VERSION(V)
```

# APPENDICI

Le seguenti appendici forniscono utili informazioni riguardanti il TI Extended BASIC.

APPENDICE A

**Elenco dei programmi dimostrativi**

| *ELEMENTI ILLUSTRATI* | *LINEE* | *DESCRIZIONE* | *PAGINA* |
|---|---|---|---|
| | 44 | Gioco CODEBREAKER | 27 |
| ACCEPT | 16 | Input di 20 nomi | 48 |
| CALL | 8 | CLEAR e subroutine personale | 55 |
| CHAR | 12 | 1. Movimento figura | 53 |
| | 7 | 2. Azzeramento caratteri | 58 |
| CHR$ | 4 | Elenco dei codici ASCII | 60 |
| CLEAR | 3 | (Semplice esempio) | 61 |
| | 3 | (Semplice esempio) | 61 |
| COINC | 10 | (Semplice esempio) | 65 |
| COS | 6 | (Semplice esempio) | 69 |
| DATA | 14 | (Semplice esempio) | 71 |
| DELETE | 2 | (Semplice esempio) | 74 |
| DISPLAY | 18 | Disegno sullo schermo | 78 |
| ERR | 5 | (Semplice esempio) | 84 |
| FOR-TO-STEP | 11 | Disegno | 87 |
| GOSUB | 24 | Probabilita' | 90 |
| GOTO | 8 | Somma da 1 fino a 100 | 91 |
| IF-THEN-ELSE | 17 | Sequenza di numeri | 96 |
| IMAGE | 12 | (Semplice esempio) | 99 |
| | 2 | (Semplice esempio) | 100 |
| INPUT | 17 | Lettera | 103 |
| INPUT (con i files) | 12 | (Semplice esempio) | 106 |
| JOYST | 5 | Movimento sprite | 108 |
| KEY | 12 | Movimento sprite | 109 |
| LINPUT | 6 | (Semplice esempio) | 113 |
| LOCATE | 6 | (Semplice esempio) | 116 |
| LOG | 8 | Logaritmo con base qualsiasi | 117 |

APPENDICE A

**Elenco dei programmi dimostrativi**

APPENDICE B

## Comandi, Istruzioni e Funzioni

---

Il seguente e' un elenco di tutti i comandi, istruzioni e funzioni del TI Extended BASIC. Prima sono elencati i comandi; se un comando puo' anche essere usato come istruzione, la lettera "S" compare alla destra del comando. I comandi che possono essere abbreviati hanno sottolineata l'abbreviazione consentita. Segue un elenco di tutte le istruzioni del TI Extended BASIC; quelli che possono essere anche usati come comandi sono seguiti dalla lettera "C". Infine, c'e' un elenco di tutte le funzioni del TI Extended BASIC.

### Comandi del TI Extended BASIC

BREAK S
BYE
CON TINUE
DELETE S
LIST
MERGE
NUM BER
OLD
RES EQUENCE
RUN S
SAVE
SIZE
TRACE S
UNBREAK S
UNTRACE S

### Istruzioni del TI Extended BASIC

ACCEPT C
CALL
CALL CHAR C
CALL CHARPAT C
CALL CHARSET C
CALL CLEAR C
CLOSE C
CALL COINC C
CALL COLOR C
DATA
DEF
CALL DELSPRITE C
DIM C
DISPLAY C
DISPLAY USING C
CALL DISTANCE C
END
CALL ERR C
FOR C
CALL GCHAR C
GOSUB
GOTO
CALL HCHAR C
IF THEN ELSE
IMAGE
CALL INIT C
INPUT
INPUT REC
CALL JOYST C
CALL KEY C
[LET] C
CALL LINK C
LINPUT
CALL LOAD C
CALL LOCATE C
CALL MAGNIFY C
CALL MOTION C
NEXT C
ON BREAK
ON ERROR
ON GOSUB
ON GOTO
ON WARNING
OPEN C
OPTION BASE
CALL PATTERN C
CALL PEEK C
CALL POSITION C
PRINT C
PRINT USING C
RANDOMIZE C
READ C
REM C
RESTORE C
RETURN
CALL SAY C
CALL SCREEN C
CALL SOUND C
CALL SPGET C
CALL SPRITE C
STOP C
SUB
SUBEND
SUBEXIT
CALL VCHAR C
CALL VERSION C

APPENDICE B

**Comandi, Istruzioni e Funzioni**

================================================================

## Funzioni del TI Extended BASIC

| | | |
|---|---|---|
| ABS | LEN | SEG$ |
| ASC | LOG | SGN |
| ATN | MAX | SIN |
| CHR$ | MIN | SQR |
| COS | PI | STR$ |
| EOF | POS | TAB |
| EXP | RES | TAN |
| INT | RND | VAL |
| | RPT$ | |

APPENDICE C

## CODICI ASCII

I seguenti caratteri predefiniti possono essere stampati o, visualizzati sullo schermo.

| CODICE ASCII | CARATTERE | CODICE ASCII | CARATTERE | CODICE ASCII | CARATTERE |
|---|---|---|---|---|---|
| 32 | (spazio) | 65 | A | 97 | A |
| 33 | ! (punto esclamativo) | 66 | B | 98 | B |
| 34 | " (virgolette) | 67 | C | 99 | C |
| 35 | # (numero o libbra) | 68 | D | 100 | D |
| 36 | $ (dollaro) | 69 | E | 101 | E |
| 37 | % (per cento) | 70 | F | 102 | F |
| 38 | & ("e" commerciale) | 71 | G | 103 | G |
| 39 | ' (apostrofo) | 72 | H | 104 | H |
| 40 | ( (parentesi aperta) | 73 | I | 105 | I |
| 41 | ) (parentesi chiusa) | 74 | J | 106 | J |
| 42 | * (asterisco) | 75 | K | 107 | K |
| 43 | + (segno più) | 76 | L | 108 | L |
| 44 | , (virgola) | 77 | M | 109 | M |
| 45 | - (segno meno) | 78 | N | 110 | N |
| 46 | . (punto) | 79 | O | 111 | O |
| 47 | / (barra obliqua) | 80 | P | 112 | P |
| 48 | 0 | 81 | Q | 113 | Q |
| 49 | 1 | 82 | R | 114 | R |
| 50 | 2 | 83 | S | 115 | S |
| 51 | 3 | 84 | T | 116 | T |
| 52 | 4 | 85 | U | 117 | U |
| 53 | 5 | 86 | V | 118 | V |
| 54 | 6 | 87 | W | 119 | W |
| 55 | 7 | 88 | X | 120 | X |
| 56 | 8 | 89 | Y | 121 | Y |
| 57 | 9 | 90 | Z | 122 | Z |
| 58 | : (due punti) | 91 | [ (parentesi quadra aperta) | 123 | { (parentesi graffa aperta) |
| 59 | ; (punto e virgola) | 92 | \ (barra obliqua inversa) | 124 | |
| 60 | < (minore) | 93 | ] (parentesi quadra chiusa) | 125 | } (parentesi graffa chiusa) |
| 61 | = (uguale) | 94 | ∧ (elevamento a potenza) | 126 | — (tilde) |
| 62 | > (maggiore) | 95 | - (trattino) | 127 | DEL (visualizzato come spazio) |
| 63 | ? (punto interrogativo) | 96 | ` (accento grave) | | |
| 64 | @ (segno at) | | | | |

I tasti relativi ai seguenti codici ASCII possono essere attivati anche mediante l'istruzione CALL KEY.

| | | | |
|---|---|---|---|
| 1 | FCTN 7 (AID=aiuto) | 3 | FCTN 1 (DELETE=elimina) |
| 4 | FCTN 2 (INSERT=inserimento) | 6 | FCTN 8 (REDO=ripete) |
| 7 | FCTN 3 (ERASE=cancella) | 8 | FCTN S (Freccia a sinis.) |
| 9 | FCTN D (Freccia a destra) | 10 | FCTN X (Freccia in basso) |
| 11 | FCTN E (Freccia in alto) | 12 | FCTN 6 (PROC'D=prosegue) |
| 13 | ENTER | 14 | FCTN 5 (BEGIN=comincia) |
| 15 | FCTN 9 (BACK=indietro) | | |

APPENDICE D

## FREQUENZE DELLE NOTE MUSICALI

La seguente tavola da' le frequenze (arrotondate all'intero) delle quattro ottave della scala musicale. Anche se quest'elenco non comprende l'intera gamma dei toni musicali che il computer puo' riprodurre, esso puo' essere d'aiuto per la programmazione di brani musicali.

| *Frequenza* | *Nota* | | *Frequenza* | *Nota* | |
|---|---|---|---|---|---|
| 110 | LA | | 440 | LA | (un'ottava sopra DO centrale |
| 117 | LA #, SI♭ | | 466 | LA #, SI♭ | |
| 123 | SI | | 494 | SI | |
| 131 | DO | (un'ottava sotto DO centrale) | 523 | DO | (un'ottava sopra DO centrale |
| 139 | DO #, RE♭ | | 534 | DO #, RE♭ | |
| 147 | RE | | 587 | RE | |
| 156 | RE #, MI♭ | | 622 | RE #, MI♭ | |
| 165 | MI | | 659 | MI | |
| 175 | FA | | 698 | FA | |
| 185 | FA #, SOL♭ | | 740 | FA #, SOL♭ | |
| 196 | SOL | | 784 | SOL | |
| 208 | SOL #, LA♭ | | 831 | SOL #, LA♭ | |
| 220 | LA | (un'ottava sotto DO centrale) | 880 | LA | (un'ottava sopra LA corista) |
| 220 | LA | (un'ottava sotto DO centrale) | 880 | LA | (un'ottava sopra LA corista) |
| 233 | LA #, SI ♭ | | 932 | LA #, SI♭ | |
| 247 | SI | | 988 | SI | |
| 262 | DO | (DO centrale) | 1047 | DO | |
| 277 | DO #, RE♭ | | 1109 | DO #, RE♭ | |
| 294 | RE | | 1175 | RE | |
| 311 | RE #, MI♭ | | 1245 | RE #, MI♭ | |
| 330 | MI | | 1319 | MI | |
| 349 | FA | | 1397 | FA | |
| 370 | FA #, SOL♭ | | 1480 | FA #, SOI ♭ | |
| 392 | SOL | | 1568 | SOI. | |
| 415 | SOL #, LA♭ | | 1661 | SOL #, LA♭ | |
| 440 | LA | (un ottava sopra DO centrale) | 1760 | LA | |

APPENDICE E

## GRUPPI DEI CARATTERI (SETS DEI CARATTERI)

---

| *SET* | *CODICI ASCII* | *SET* | *CODICI ASCII* |
|---|---|---|---|
| 0 | 30-31 | | |
| 1 | 32-39 | 8 | 88-95 |
| 2 | 40-47 | 9 | 96-103 |
| 3 | 48-55 | 10 | 104-111 |
| 4 | 56-63 | 11 | 112-119 |
| 5 | 64-71 | 12 | 120-126 |
| 6 | 72-79 | 13 | 128-135 |
| 7 | 80-87 | 14 | 136-143 |

APPENDICE F

## IDENTIFICATORE DI SAGOMA
## TAVOLA DI CONVERSIONE

---

| *Blocchi* | *CODICE BINARIO (0=off; 1=on)* | *CODICE ESADECIMALE* |
|---|---|---|
| | 0000 | 0 |
| | 0001 | 1 |
| | 0010 | 2 |
| | 0011 | 3 |
| | 0100 | 4 |
| | 0101 | 5 |
| | 0110 | 6 |
| | 0111 | 7 |
| | 1000 | 8 |
| | 1001 | 9 |
| | 1010 | A |
| | 1011 | B |
| | 1100 | C |
| | 1101 | D |
| | 1110 | E |
| | 1111 | F |

APPENDICE G

## CODICI DEI COLORI

| *COLORE* | *CODICE* |
|---|---|
| Trasparente | 1 |
| Nero | 2 |
| Verde | 3 |
| Verde chiaro | 4 |
| Blu scuro | 5 |
| Blu chiaro | 6 |
| Rosso scuro | 7 |
| Ciano | 8 |
| Rosso | 9 |
| Rosso chiaro | 10 |
| Giallo scuro | 11 |
| Giallo chiaro | 12 |
| Verde scuro | 13 |
| Magenta | 14 |
| Grigio | 15 |
| Bianco | 16 |

PENDICE H

**MBINAZIONI DEI COLORI**

==========================================================

seguenti combinazioni sono le piu' idonee per ottenere i
ontrasti migliori tra colore dello sfondo e colore dei
aratteri.

PRIMA SCELTA

| | | | |
|---|---|---|---|
| ,8 | nero su ciano | 2,13 | nero su verde scuro |
| ,7 | nero su rosso scuro | 2,15 | nero su grigio |
| ,6 | nero su blu chiaro | 2,14 | nero su magenta |
| ,3 | nero su verde | 2,9 | nero su rosso |
| ,8 | blu scuro su ciano | 5,15 | blu scuro su grigio |
| ,6 | blu scuro su blu chiaro | 5,4 | blu scuro su verde chiaro |
| ,14 | blu scuro su magenta | 5,16 | blu scuro su bianco |
| 3,8 | verde scuro su ciano | 13,11 | verde scuro su giallo scuro |
| 3,15 | verde scuro su grigio | 13,4 | verde scuro su verde chiaro |
| 3,12 | verde scuro su giallo chiaro | 13,3 | verde scuro su verde |
| ,15 | rosso scuro su grigio | 7,10 | rosso scuro su rosso chiaro |
| ,12 | rosso scuro su giallo chiaro | 14,2 | magenta su rosso chiaro |
| ,12 | verde su giallo chiaro | 3,15 | verde su bianco |

SECONDA SCELTA

| | | | |
|---|---|---|---|
| ,5 | nero su blu scuro | 2,11 | nero su giallo scuro |
| ,4 | nero su verde chiaro | 2,10 | nero su rosso chiaro |
| ,12 | nero su giallo chiaro | 13,10 | verde scuro su rosso chiaro |
| 3,16 | verde scuro su bianco | 7,16 | rosso scuro su bianco |
| ,15 | blu chiaro su grigio | 6,4 | blu chiaro su verde chiaro |
| ,16 | blu chiaro su bianco | 4,16 | verde chiaro su bianco |

TERZA SCELTA

| | | | |
|---|---|---|---|
| ,16 | nero su bianco | 5,12 | blu scuro su giallo chiaro |
| ,9 | rosso scuro su rosso | 4,12 | verde chiaro su giallo chiaro |
| 4,15 | magenta su grigio | 14,16 | magenta su bianco |
| ,11 | verde su giallo scuro | 3,15 | verde su grigio |
| ,15 | rosso su grigio | 9,10 | rosso su rosso chiaro |
| ,12 | rosso su giallo chiaro | 9,16 | rosso su bianco |
| 6,7 | bianco su rosso scuro | | |

QUARTA SCELTA

| | | | |
|---|---|---|---|
| ,2 | ciano su nero | 8,16 | ciano su bianco |
| ,2 | rosso scuro su nero | 7,4 | rosso scuro su verde chiaro |
| 5,16 | grigio su bianco | 5,2 | blu chiaro su nero |
| ,2 | verde chiaro su nero | 10,2 | rosso chiaro su nero |
| 0,16 | rosso chiaro su bianco | 14,12 | magenta su giallo chiaro |
| ,4 | rosso su verde chiaro | 16,6 | bianco su blu chiaro |

APPENDICE I

## SUDDIVISIONE DELLA TASTIERA

Unità tastiera 1 — Unità tastiera 2

APPENDICE J

## CODICI DEI CARATTERI PER LA SUDDIVISIONE DELLA TASTIERA

| CODICI | TASTI* | CODICI | TASTI* |
|---|---|---|---|
| 0 | X,M | 10 | 5,0 |
| 1 | A,H | 11 | T,P |
| 2 | S,J | 12 | F,L |
| 3 | D,K | 13 | V,.(punto) |
| 4 | W,U | 14 | C,,(virgola) |
| 5 | E,I | 15 | Z,N |
| 6 | R,O | 16 | B,/(barra obliqua) |
| 7 | 2,7 | 17 | G; (punto e virgola) |
| 8 | 3,8 | 18 | Q,Y |
| 9 | 4,9 | 19 | 1,6 |

*** Notate che il primo dei due tasti è sulla sinistra della tastiera ed il secondo è sulla destra.**

APPENDICE K

**FUNZIONI MATEMATICHE**

==================================================================

Le seguenti funzioni matematiche possono essere definite mediante la funzione DEF.

| *Funzione* | *Istruzione TI Extended BASIC* |
|---|---|
| Secante | DEF SEC(X)=1/COS(X) |
| Cosecante | DEF CSC(X)=1/SIN(X) |
| Cotangente | DEF COT(X)=1/TAN(X) |
| Inverso seno | DEF ARCSIN(X)=ATN(X/SQR(1-X*X)) |
| Inverso coseno | DEF ARCCOS(X)=-ATN(X/SQR(1-X*X))+PI/2 |
| Inverso secante | DEF ARCSEC(X)=ATN(SQR(X*X-1))+(SGN(X)-1)*PI/2 |
| Inverso cosecante | DEF ARCCSC(X)=ATN(1/SQR(X*X-1))+(SGN(X)-1)*PI/2 |
| Inverso cotangente | DEF ARCCOT(X)=PI/2-ATN(X) or =PI/2+ATN(-X) |
| Seno iperbolico | DEF SINH(X)=(EXP(X)-EXP(-X))/2 |
| Coseno iperbolico | DEF COSH(X)=(EXP(X)+EXP(-X))/2 |
| Tangente iperbolica | DEF TANH(X)=-2*EXP(-X)/(EXP(X)+EXP(-X))+1 |
| Secante iperbolica | DEF SECH=2/(EXP(X)+EXP(-X)) |
| Cosecante iperbolica | DEF CSCH=2/(EXP(X)-EXP(-X)) |
| Cotangente iperbolica | DEF COTH(X)=2*EXP(-X)/(EXP(X)-EXP(-X))+1 |
| Inverso seno iperbolico | DEF ARCSINH(X)=LOG(X+SQR(X*X+1)) |
| Inverso coseno iperbolico | DEF ARCCOSH(X)=LOG(X+SQR(X*X-1)) |
| Inverso tangente iperbolica | DEF ARCTANH(X)=LOG((1+X)/(1-X))/2 |
| Inverso secante iperbolica | DEF ARCSECH(X)=LOG((1+SQR(1-X*X))/X) |
| Inverso cosecante iperbolica | DEF ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X) |
| Inverso cotangente iperbolica | DEF ARCCOTH(X)=LOG((X+1)/(X-1))/2 |

APPENDICE L

# ELENCO DEI VOCABOLI DEL SINTETIZZATORE VOCALE

Il seguente e' un elenco di tutte le lettere, numeri, parole e frasi alle quali si puo' accedere mediante CALL SAY e CALL SPGET. Vedere l'appendice M per le istruzioni su come aggiungervi i suffissi.

- (NEGATIVE)
+ (POSITIVE)
. (POINT)
0
1
2
3
4
5
6
7
8
9
A (a)
A1 (ə)
ABOUT
AFTER
AGAIN
ALL
AM
AN
AND
ANSWER
ANY
ARE
AS
ASSUME
AT
B
BACK
BASE
BE
BETWEEN
BLACK
BLUE
BOTH
BOTTOM
BUT
BUY
BY
BYE
C
CAN
CASSETTE
CENTER
CHECK
CHOICE
CLEAR
COLOR
COME
COMES
COMMA
COMMAND
COMPLETE
COMPLETED
COMPUTER
CONNECTED
CONSOLE
CORRECT
COURSE
CYAN
D
DATA
DECIDE
DEVICE
DID
DIFFERENT
DISKETTE
DO
DOES
DOING
DONE
DOUBLE
DOWN
DRAW
DRAWING
E
EACH
EIGHT
EIGHTY
ELEVEN
ELSE
END
ENDS
ENTER
ERROR
EXACTLY
EYE
F
FIFTEEN
FIFTY
FIGURE
FIND
FINE
FINISH
FINISHED
FIRST
FIT
FIVE
FOR
FORTY
FOUR
FOURTEEN
FOURTH
FROM
FRONT
G
GAMES
GET
GETTING
GIVE
GIVES
GO
GOES
GOING
GOOD
GOOD WORK
GOODBYE
GOT
GRAY
GREEN
GUESS
H
HAD
HAND
HANDHELD UNIT
HAS
HAVE
HEAD
HEAR
HELLO
HELP

APPENDICE L

## ELENCO DEI VOCABOLI DEL SINTETIZZATORE VOCALE

HERE
HIGHER
HIT
HOME
HOW
HUNDRED
HURRY
I
I WIN
IF
IN
INCH
INCHES
INSTRUCTION
INSTRUCTIONS
IS
IT
J
JOYSTICK
JUST
K
KEY
KEYBOARD
KNOW
L
LARGE
LARGER
LARGEST
LAST
LEARN
LEFT
LESS
LET
LIKE
LIKES
LINE
LOAD
LONG
LOOK
LOOKS
LOWER
M
MADE
MAGENTA
MAKE
ME
MEAN
MEMORY
MESSAGE
MESSAGES
MIDDLE
MIGHT
MODULE
MORE
MOST
MOVE
MUST
N
NAME
NEAR
NEED
NEGATIVE
NEXT
NICE TRY
NINE
NINETY
NO
NOT
NOW
NUMBER
O
OF
OFF
OH
ON
ONE
ONLY
OR
ORDER
OTHER
OUT
OVER
P
PART
PARTNER
PARTS
PERIOD
PLAY
PLAYS
PLEASE
POINT
POSITION
POSITIVE
PRESS
PRINT
PRINTER
PROBLEM
PROBLEMS
PROGRAM
PUT
PUTTING
Q
R
RANDOMLY
READ (read)
READ1 (red)
READY TO START
RECORDER
RED
REFER
REMEMBER
RETURN
REWIND
RIGHT
ROUND
S
SAID
SAVE
SAY
SAYS
SCREEN
SECOND
SEE
SEES
SET
SEVEN
SEVENTY
SHAPE
SHAPES
SHIFT
SHORT
SHORTER
SHOULD
SIDE
SIDES
SIX
SIXTY
SMALL
SMALLER
SMALLEST
SO
SOME
SORRY

APPENDICE L

ELENCO DEI VOCABOLI DEL
SINTETIZZATORE VOCALE

SPACE
SPACES
SPELL
SQUARE
START
STEP
STOP
SUM
SUPPOSED
SUPPOSED TO
SURE
T
TAKE
TEEN
TELL
TEN
TEXAS INSTRUMENTS
THAN
THAT
THAT IS INCORRECT
THAT IS RIGHT
THE (the)
THE1 (thə)
THEIR
THEN
THERE
THESE
THEY
THING
THINGS
THINK
THIRD
THIRTEEN
THIRTY
THIS
THREE
THREW
THROUGH
TIME
TO
TOGETHER
TONE
TOO
TOP
TRY
TRY AGAIN
TURN
TWELVE
TWENTY
TWO
TYPE
U
UHOH
UNDER
UNDERSTAND
UNTIL
UP
UPPER
USE
V
VARY
VERY
W
WAIT
WANT
WANTS
WAY
WE
WEIGH
WEIGHT
WELL
WERE
WHAT
WHAT WAS THAT
WHEN
WHERE
WHICH
WHITE
WHO
WHY
WILL
WITH
WON
WORD
WORDS
WORK
WORKING
WRITE
X
Y
YELLOW
YES
YET
YOU
YOU WIN
YOUR
Z
ZERO

APPENDICE M

## AGGIUNTA DEI SUFFISSI AI VOCABOLI DEL SINTETIZZATORE VOCALE

Questa appendice descrive come aggiungere ING, S, ED a ciascuna parola residente nel vocabolario del sintetizzatore vocale.

Il codice per una parola e' prima letto utilizzando SPGET. Il codice consiste in un numero di caratteri, ciascuno dei quali dice al sintetizzatore quanto e' lunga la parola. Quindi mediante quanto specificato nel sottoprogramma listato qui di seguito, possono essere aggiunti codici per dare il suono di un suffisso.

Spesso le parole sono state schematizzate all'interno del sintetizzatore vocale per renderne il suono piu' naturale possibile. Aggiungendo i suffissi queste possono essere alterate.

Il seguente programma vi permette di inserire una parola, e provando diversi valori di troncamento rendere il suono del suffisso simile al suono naturale della parola. Il sottoprogramma DEFING (dalla linea 1000 alla 1130), DEFS1(dalla 2000 alla 2100), DEFS2(dalla 3000 alla 3090), DEFS3(dalla 4000 alla 4120), DEFED1(dalla 5000 alla 5070), DEFED2(dalla 6000 alla 6110), DEFED3(dalla 7000 alla 7130) e MENU(dalla 10000 alla 10120) dovrebbero essere salvati di volta in volta con l'opzione MERGE. (Il sottoprogramma MENU e' lo stesso usato nel programma dimostrativo dell'istruzione SUB.) Se desiderate potete usare differenti numeri di linea. Ciascuno di questi sottoprogrammi (eccetto MENU) definisce un suffisso.

DEFING definisce il suono ING. DEFS1 definisce il suono S come si pronuncia alla fine di "cats". DEFS2 definisce la S come in "cads". DEFS3 definisce la S come in "wishes". DEFED1 definisce il suono ED come si pronuncia alla fine della parola "passed". DEFD2 lo definisce come in "caused". DEFED3 lo definisce come in "heated".

Eseguendo il programma inserire uno 0 per il valore di troncamento per poter finire la sequenza.

```
100 REM ******************
110 REM REQUIRES MERGE OF:
120 REM MENU (LINES 10000 THROUGH 10120)
130 REM DEFING (LINES 1000 THROUGH 1130)
140 REM DEFS1 (LINES 2000 THROUGH 2100)
150 REM DEFS2 (LINES 3000 THROUGH 3090)
160 REM DEFS3 (LINES 4000 THROUGH 4120)
170 REM DEFED1 (LINES 5000 THROUGH 5070)
180 REM DEFED2 (LINES 6000 THROUGH 6110)
190 REM DEFED3 (LINES 7000 THROUGH 7130)
200 REM ******************
210 CALL CLEAR
220 PRINT "THIS PROGRAM IS USED TO"
```

ELENCO DEI VOCABOLI DEL
SINTETIZZATORE VOCALE

```
230 PRINT "FIND THE PROPER TRUNCATION"
240 PRINT "VALUE FOR ADDING SUFFIXES"
250 PRINT "TO SPEECH WORDS.": :
260 FOR DELAY=1 TO 300::NEXT DELAY
270 PRINT "CHOOSE WHICH SUFFIX YOU"
280 PRINT "WISH TO ADD.": :
290 FOR DELAY=1 TO 200::NEXT DELAY
300 CALL MENU(8,CHOICE)
310 DATA 'ING','S' AS IN CATS,'S' AS IN CADS,'S' AS IN WISHES,
'ED' AS IN PASSED,'ED' AS IN CAUSED,'ED' AS IN HEATED,END
320 IF CHOICE=0 OR CHOICE=8 THEN STOP
330 INPUT "WHAT IS THE WORD? ":WORD$
340 ON CHOICE GOTO 350,370,390,410,430,450,470
350 CALL DEFING(D$)
360 GOTO 480
370 CALL DEFS1(D$)!CATS
380 GOTO 480
390 CALL DEFS2(D$)!CADS
400 GOTO 480
410 CALL DEFS3(D$)!WISHES
420 GOTO 480
430 CALL DEFED1(D$)!PASSED
440 GOTO 480
450 CALL DEFED2(D$)!CAUSED
460 GOTO 480
470 CALL DEFED3(D$)!HEATED
480 REM TRY VALUES
490 CALL CLEAR
500 INPUT "TRUNCATE HOW MANY BYTES? ":L
510 IF L=0 THEN 300
520 CALL SPGET(WORD$,B$)
530 L=LEN(B$)-L-3
540 C$=SEG$(B$,1,2)&CHR$(L)&SEG$(B$,4,L)
550 CALL SAY(,C$&D$)
560 GOTO 500
```

APPENDICE M

## ELENCO DEI VOCABOLI DEL SINTETIZZATORE VOCALE

I dati sono stati espressi in istruzioni DATA abbreviate per permettere un piu' facile ingresso. Cio' deve essere tenuto presente per creare un programma piu' corto.

```
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,184,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR I=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT I
1130 SUBEND

2000 SUB DEFS1(A$)!CATS
2010 DATA 96,0,26
2020 DATA 14,56,130,204,0
2030 DATA 223,177,26,224,103
2040 DATA 85,3,252,106,106
2050 DATA 128,95,44,4,240
2060 DATA 35,11,2,126,16,121
2070 RESTORE 2010
2080 A$=""
2090 FOR I=1 TO 29::READ A::A$=A$&CHR$(A)::NEXT I
2100 SUBEND

3000 SUB DEFS2(A$)!CADS
3010 DATA 96,0,17
3020 DATA 161,253,158,217
3030 DATA 168,213,198,86,0
3040 DATA 223,153,75,128,0
3050 DATA 95,139,62
3060 RESTORE 3010
3070 A$=""
3080 FOR I=1 TO 20::READ A::A$=A$&CHR$(A)::NEXT I
3090 SUBEND
```

APPENDICE M

# ELENCO DEI VOCABOLI DEL SINTETIZZATORE VOCALE

```
4000 SUB DEFS3(A$)!WISHES
4010 DATA 96,0,34
4020 DATA 173,233,33,84,12
4030 DATA 242,205,166,55,173
4040 DATA 93,222,68,197,188
4050 DATA 134,238,123,102
4060 DATA 163,86,27,59,1,124
4070 DATA 103,46,1,2,124,45
4080 DATA 138,129,7
4090 RESTORE 4010
4100 A$=""
4110 FOR I=1 TO 37::READ A::A$=A$&CHR$(A)::NEXT I
4120 SUBEND

5000 SUB DEFED1(A$)!PASSED
5010 DATA 96,0,10
5020 DATA 0,224,128,37
5030 DATA 204,37,240,0,0,0
5040 RESTORE 5010
5050 A$=""
5060 FOR I=1 TO 13::READ A::A$=A$&CHR$(A)::NEXT I
5070 SUBEND

6000 SUB DEFED2(A$)!CAUSED
6010 DATA 96,0,26
6020 DATA 172,163,214,59,35
6030 DATA 109,170,174,68,21
6040 DATA 22,201,220,250,24
6050 DATA 69,148,162,166,234
6060 DATA 75,84,97,145,204
6070 DATA 15
6080 RESTORE 6010
6090 A$=""
6100 FOR I=1 TO 29::READ A::A$=A$&CHR$(A)::NEXT I
6110 SUBEND
```

APPENDICE M

## ELENCO DEI VOCABOLI DEL SINTETIZZATORE VOCALE

==================================================================

```
7000 SUB DEFED3(A$)!HEATED
7010 DATA 96,0,36
7020 DATA 173,233,33,84,12
7030 DATA 242,205,166,183
7040 DATA 172,163,214,59,35
7050 DATA 109,170,174,68,21
7060 DATA 22,201,92,250,24
7070 DATA 69,148,162,38,235
7080 DATA 75,84,97,145,204
7090 DATA 178,127
7100 RESTORE 7010
7110 A$=""
7120 FOR I=1 TO 39::READ A::A$=A$&CHR$(A)::NEXT I
7130 SUBEND

10000 SUB MENU(COUNT,CHOICE)
10010 CALL CLEAR
10020 IF COUNT>22 THEN PRINT "TOO MANY ITEMS" :: CHOICE=0 :: SUBEXIT
10030 RESTORE
10040 FOR I=1 TO COUNT
10050 READ TEMP$
10060 TEMP$=SEG$(TEMP$,1,25)
10070 DISPLAY AT(I,1):I;TEMP$
10080 NEXT I
10090 DISPLAY AT(I+1,1):"YOUR CHOICE: 1"
10100 ACCEPT AT(I+1,14)BEEP VALIDATE(DIGIT)SIZE(-2):CHOICE
10110 IF CHOICE<1 OR CHOICE>COUNT THEN 10100
10120 SUBEND
```

APPENDICE M

## ELENCO DEI VOCABOLI DEL SINTETIZZATORE VOCALE

Potete usare i sottoprogrammi in ciascun programma una volta che avete determinato il numero dei bytes da troncare. Il seguente programma usa il sottoprogramma DEFING dalla linea 1000 alla 1130 facendo pronunciare al computer la parola DRAWING usando DRAW piu' il suffisso ING. Osservare che cio' che e' stato trovato, DRAW, potrebbe essere troncato da 41 caratteri al fine di produrre il suono piu' naturale di DRAWING. Il sottoprogramma DEFING dalla linea 1000 alla 1130 e' il programma che e' stato salvato con l'opzione MERGE.

```
100 CALL DEFING(ING$)
110 CALL SPGET("DRAW",DRAW$)
120 L=LEN(DRAW$)-3-41! 3 BYTES OF SPEECH OVERHEAD, 41 BYTES TRUNCATED
130 DRAW$=SEG$(DRAW$,1,2)&CHR$(L)&SEG$(DRAW$,4,L)
140 CALL SAY("WE ARE",DRAW$&ING$,"A1 SCREEN")
150 GOTO 140
1000 SUB DEFING(A$)
1010 DATA 96,0,52,174,30,65
1020 DATA 21,186,90,247,122,214
1030 DATA 179,95,77,13,202,50
1040 DATA 153,120,117,57,40,248
1050 DATA 133,173,209,25,39,85
1060 DATA 225,54,75,167,29,77
1070 DATA 105,91,44,157,118,180
1080 DATA 169,97,161,117,218,25
1090 DATA 119,164,227,222,249,238,1
1100 RESTORE 1010
1110 A$=""
1120 FOR I=1 TO 55::READ A::A$=A$&CHR$(A)::NEXT I
1130 SUBEND
```

APPENDICE N

**ERRORI**

=================================================================

Quanto segue e' un elenco di tutti i messaggi di errore disponibili con il TI Extended BASIC. Il primo elenco e' in ordine alfabetico per tipo di messaggio, il secondo e' ordinato in base ai numeri dei codici degli errori che sono restituiti usando CALL ERR. Se l'errore avviene durante l'esecuzione del programma, oltre al messaggio di errore viene specificato anche il numero della linea nella quale si e' verificato l'errore.

**Per ordine di messaggio**

| # | *Messaggio* | *Errori possibili* |
|---|---|---|

74 **BAD ARGUMENT**

* Valore incorretto in ASC, ATN, COS, EXP, INT, LOG, SIN, SOUND, SQR, TAN o VAL.
* E' stato specificato un elemento di matrice in una istruzione SUB.
* Primo parametro errato o troppi parametri in una istruzione LINK.

61 **BAD LINE NUMBER**

* Numero di linea minore di 1 o maggiore di 32767.
* Numero di linea omesso.
* Numero di linea prodotto da RES, al di fuori della gamma compresa tra 1 e 32767.

57 **BAD SUBSCRIPT**

* Indice di una matrice troppo grande o piccolo.
* Indice incorretto in DIM.

79 **BAD VALUE**

* Valore errato dato in AND, CHAR, CHR$, CLOSE, EOF, FOR, GOSUB, GOTO, HCHAR, INPUT, MOTION, NOT, OR, POS, PRINT, PRINT USING, REC, RESTORE, RPT$, SEG$, SIZE, VCHAR o XOR.
* Indice di matrice superiore a 32767.
* Numero di file maggiore di 255 o minore di zero.
* Sono stati specificati piuo di tre toni ed un rumore in SOUND.
* Un valore trasferito ad un sottoprogramma non e' accettabile nel sottoprogramma. Per esempio, la velocita' di uno sprite minore di -128 o un valore riferito al codice di un carattere superiore a 143.
* Il valore dato in ON...GOTO o in ON...GOSUB e' superiore al numero delle linee specificate.
* Posizione incorretta specificata dopo la clausola AT nell'isruzione ACCEPT o DISPLAY.

67 **CAN'T CONTINUE**

* E' stata effettuata una correzione in un programma interrotto da un breakpoint.
* Non e' stata interrotta l'esecuzione di alcun programma.

69 **COMMAND ILLEGAL IN PROGRAM**

* E' stato usato in un programma un comando BYE, CON, LIST, MERGE, NEW, NUM, OLD, RES o SAVE.

**ERRORI**

==================================================================

84 **DATA ERROR**
* READ o RESTORE con dati non presenti o con una stringa dove ci si aspetta un valore numerico.
* Il numero di linea dopo RESTORE e' superiore al piu' alto numero di linea del programma.
* Errore nel file oggetto in LOAD.

109 **FILE ERROR**
* Errato tipo dei dati letti con un'istruzione READ.
* Tentativo di usare CLOSE, EOF, INPUT, OPEN, PRINT, PRINT USING, REC o RESTORE con un file che non esiste o che non ha gli attributi adatti.

44 **FOR-NEXT NESTING**
* (Cicli nidificati) Le istruzioni FOR e NEXT dei loops non si allineano appropriatamente.
* Istruzione NEXT mancante.

130 **I/O ERROR**
* E' stato riconosciuto un errore nel tentativo di eseguire CLOSE, DELETE, LOAD, MERGE, OLD, OPEN, RUN o SAVE.
* Memoria insufficiente a listare un programma.

16 **ILLEGAL AFTER SUBPROGRAM**
* Dopo SUBEND sono validi solo END, REM, SUB.

36 **IMAGE ERROR**
* Si e' verificato un errore nell'uso di DISPLAY USING, IMAGE o PRINT USING.
* Piu' di 10(nel formato E) o di 14(nel formato numerico) cifre significative nel campo relativo.
* La stringa del formato IMAGE e' piu' lunga di 254 caratteri.

28 **IMPROPERLY USED NAME**
* Nome illegale di variabile in CALL, DEF o DIM.
* Parola riservata del TI Extended BASIC in LET.
* Variabile con indice o una stringa e' usata in FOR
* Matrice usata con numero errato di dimensionamento
* Si usa un nome di variabile diverso da quello precedentemente assegnato. Il nome di variabile puo' essere una matrice, numerica o stringa, o il il nome di una funzione definita.
* Matrice dimensionata due volte.
* E' stato messo il nome di una funzione definita sulla sinistra del segno di uguale in un'istruzion di asseganzione.
* Viene usata due volte la stessa variabile nell'elenco dei parametri di un'istruzione SUB.

\PPENDICE N

ERRORI

===========================================================================

31 **INCORRECT ARGUMENT LIST**
* Disuguaglianza di argomenti in CALL e SUB.

33 **INPUT ERROR**
* Errore verificatosi in una INPUT.

50 **LINE NOT FOUND**
* Riscontrato un errato numero di linea in BREAK, GOSUB, GOTO, ON ERROR, RUN o UNBREAK, o dopo THEN o ELSE.
* Non esiste la linea richiamata in editing.

52 **LINE TOO LONG**
* Linea troppo lunga per essere inserita nel programma.

39 **MEMORY FULL**
* Programma troppo grande per poter eseguire: DEF, DELETE, DIM, GOSUB, LET, LOAD, ON...GOSUB, OPEN o SUB.
* Programma troppo grande per poter aggiungere o inserire o sostituire una linea, o per calcolare un'espressione.

39 **MISSING SUBEND**
* Manca la SUBEND in un sottoprogramma.

37 **MUST BE IN SUBPROGRAM**
* Sono state specificate SUBEND o SUBEXIT al di fuori di un sottoprogramma.

19 **NAME TOO LONG**
* Nome di variabile o di sottoprogramma piu' lungo di 15 caratteri.

43 **NEXT WITHOUT FOR**
* Manca l'istruzione FOR. NEXT prima di FOR, FOR-NEXT sistemati in modo incorretto o ramificati in un ciclo FOR-NEXT.

78 **NO PROGRAM PRESENT**
* Dopo i comandi LIST, RESEQUENCE, RESTORE, RUN o SAVE se non esiste alcun programma in memoria.

10 **NUMERIC OVERFLOW**
* Il numero ottenuto come risultato dopo le operazioni *, +, -, /, o dopo ACCEPT, ATN, COS, EXP, INPUT, INT, LOG, SIN, SQR, TAN o VAL, e' troppo grande o troppo piccolo.

70 **ONLY LEGAL IN A PROGRAM**
* Una delle seguenti istruzioni e' stata usata come comando:DEF, GOSUB, GOTO, IF, IMAGE, INPUT, ON BREAK, ON ERROR, ON...GOSUB, ON...GOTO, ON WARNING, OPTION BASE, RETURN, SUB, SUBEND o SUBEXIT.

**ERRORI**

========================================================================

25 **OPTION BASE ERROR**
- OPTION BASE eseguito per piu' di una volta o con valore diverso da 1 o da zero.

97 **PROTECTION VIOLATION**
- Tentativo di salvare, listare o correggere un programma protetto.

48 **RECURSIVE SUBPROGRAM CALL**
- Sottoprogramma che richiama se stesso direttamente o indirettamente.

51 **RETURN WITHOUT GOSUB**
- RETURN senza GOSUB o un errore trattato dalla precedente esecuzione di un'istruzione ON ERROR.

56 **SPEECH STRING TOO LONG**
- La stringa vocale restituita da SPGET e' piu' lunga di 255 caratteri.

40 **STACK OVERFLOW**
- Troppe serie di parentesi.
- Memoria insufficiente per calcolare un'espressione o per assegnare un valore.

54 **STRING TRUNCATED**
- Una stringa creata con RPT$, con la concatenazione ( operatore ampersand ) o una funzione definita dall'utente e' piu' lunga di 255 caratteri.
- La lunghezza di un'espressione di stringa nella clausola VALIDATE e' maggiore di 254 caratteri.

24 **STRING NUMBER MISMATCH**
- E' stata data una stringa dove ci si aspettava un numero o viceversa in una funzione TI Extended BASIC o in un sottoprogramma.
- Assegnazione di una di stringa ad un valore numerico o viceversa.
- Tentativo di concatenare ( operatore Ampersand ) un numero.
- Si usa una stringa come indice.

135 **SUBPROGRAM NOT FOUND**
- Il sottoprogramma chiamato non esiste oppure un sottoprogramma in linguaggio Assembly chiamato con LINK non e' stato caricato.

APPENDICE N

ERRORI

==========================================================================

**14 SYNTAX ERROR**

* Errore di sintassi dovuto alla mancanza di virgole o parentesi, parametri in ordine errato, mancanza di parametri, mancanza di parole chiave, o parole-chiave scritte in modo incorretto o non nel giusto ordine.
* DATA o IMAGE non compaiono come prime o uniche istruzioni di una linea.
* Voci dopo la parentesi finale ")".
* Mancanza del "#" nelle istruzioni SPRITE.
* Mancanza di ENTER, del simbolo di commento (!) o del simbolo separatore di istruzione (::).
* Mancanza di THEN dopo IF.
* Mancanza di TO dopo FOR.
* Non c'e' niente dopo CALL, SUB, FOR, THEN o ELSE.
* Due E in una costante numerica.
* Errato elenco di parametri in un sottoprogramma definito dall'utente.
* Ingresso o uscita da un sottoprogramma con GOTO, GOSUB, ON ERROR etc..
* Chiamata di INIT senza aver acceso l'espansione di memoria collegata.
* Chiamata di LINK o LOAD senza la prioritaria INIT.
* Usare una costante dove e' richiesta una variabile.
* Matrice definita con piu' di sette dimensioni.

**17 UNMATCHED QUOTES**

* Disuguaglianza di apici in una linea in ingresso.

**20 UNRECOGNIZED CHARACTER**

* Un carattere non riconosciuto come un punto interrogativo o un simbolo di percento.
* Campo errato in un file oggetto al quale si accede mediante LOAD.

APPENDICE N

## ERRORI

---

### In ordine di #

| # | *Messaggio* |
|---|---|
| 10 | NUMERIC OVERFLOW |
| 14 | SYNTAX ERROR |
| 16 | ILLEGAL AFTER SUBPROGRAM |
| 17 | UNMATCHED QUOTES |
| 19 | NAME TOO LONG |
| 20 | UNRECOGNIZED CHARACTER |
| 24 | STRING-NUMBER MISMATCH |
| 25 | OPTION BASE ERROR |
| 28 | IMPROPERLY USED NAME |
| 36 | IMAGE ERROR |
| 39 | MEMORY FULL |
| 40 | STACK OVERFLOW |
| 43 | NEXT WITHOUT FOR |
| 44 | FOR-NEXT NESTING |
| 47 | MUST BE IN SUBPROGRAM |
| 48 | RECURSIVE SUBPROGRAM CALL |
| 49 | MISSING SUBEND |
| 51 | RETURN WITHOUT GOSUB |
| 54 | STRING TRUNCATED |
| 56 | SPEECH STRING TOO LONG |
| 57 | BAD SUBSCRIPT |
| 60 | LINE NOT FOUND |
| 61 | BAD LINE NUMBER |
| 62 | LINE TOO LONG |
| 67 | CAN'T CONTINUE |
| 69 | COMMAND ILLEGAL IN PROGRAM |
| 70 | ONLY LEGAL IN A PROGRAM |
| 74 | BAD ARGUMENT |
| 78 | NO PROGRAM PRESENT |
| 79 | BAD VALUE |
| 81 | INCORRECT ARGUMENT LIST |
| 83 | INPUT ERROR |
| 84 | DATA ERROR |
| 97 | PROTECTION VIOLATION |
| 109 | FILE ERROR |
| 130 | I/O ERROR |
| 135 | SUBPROGRAM NOT FOUND |

## AVVERTENZE IMPORTANTI - USO GENERALE

=================================================================

**Ripetizione automatica**
Usando il TI Extended BASIC, tenendo premuto un tasto per piu' di un secondo, il carattere corrispondente verra' ripetuto sullo schermo finche' questo non viene rilasciato.

**Tasti di controllo**

Il TI 994/A possiede anche i caratteri di controllo che sono usati prevalentemente per le telecomunicazioni. Per inserire una di queste funzioni, premere contemporaneamente il tasto **CTRL** e il tasto della lettera, numero o simbolo relativo alla funzione desiderata.

**Istruzione DATA**

Il computer legge ciascuna informazione inserita dopo l'istruzione DATA come una parte dell'istruzione stessa. Tuttavia in una linea di programma multi-istruzione un'istruzione DATA non deve essere seguita da un'altra istruzione.

**PRE-SCAN —!@P— and!@P+**

Dopo aver inserito il RUN per far partire un programma, si puo' osservare una pausa prima che questo cominci. Questa pausa e' dovuta alla "prescansione", ovvero alla verifica effettuata dal computer sul programma. In questa fase viene riservato spazio in memoria per le variabili, per le matrici e i dati. Poi attraverso ciascuna istruzione il computer procede a perfezionare le appropriate funzioni e a stabilire i valori delle variabili. Poiche' il tempo richiesto dal pre-scan dipende dalla lunghezza del programma e' possibile diminuire questa pausa in particolare se il programma e' abbastanza lungo.
I nuovi comandi di pre-scan del TI Extended BASIC **!@P-** e **!@P+** , permettono di controllare quali istruzioni saranno esaminate e quali no. Poiche' il proposito del pre-scan e' quello di riservare spazio in memoria per le variabili, soltanto alcune istruzioni che contengono il primo riferimento alle variabili necessiteranno di essere verificate. Tuttavia molte altre istuzioni non richiedono il pre-scan.
Un'accurata pianificazione del programma e' richiesta per diminuire le istruzioni che necessitano del pre-scan. Quando alcuni tipi di istruzioni (come appresso spiegato) sono usate nel programma, le procedure elencate di seguito dovranno essere incluse nel pre-scan.

Inserire la prima istruzione data senza il pre-scan.
Includere le prime variabili o matrici usate. (Inserire anche l'istruzione OPTION BASE se e' usata nel programma.)
Includere il primo riferimento a ciascuna istruzione CALL di ciascun sottoprogramma.
Includere tutte le istruzioni DEF delle funzioni autodefinite.
Includere tutte le istruzioni SUB e SUBEND.

Osservare che una variabile in un sottoprogramma (SUB) definito dall'utente, e' considerata diversamente dalle altre variabili con lo stesso nome e lo stesso valore inserite in qualsiasi altra parte del programma. Tuttavia, nel pre-scan devono essere incluse tutte le variabili usate nei sottoprogrammi.

Per usare l'opzione di pre-scan, assicurarsi prima che il programma completo venga eseguito correttamente. Quindi all'inizio di un gruppo di istruzioni di funzione, usare il comando !@P- per disattivare il pre-scan. Le istruzioni successive non saranno verificate, facendo in modo che l'escuzione del programma cominci piu' rapidamente. Ogni istruzione relativa ai nomi di variabili (alle quali non ci sia precedentemente riferiti durante il pre-scan) causa un SINTAX ERROR se il pre-scan e' disattivato. Osservare che !@P- non puo' essere seguito da un'altra istruzione.

Per riattivare il pre-scan digitare semplicemente il comando !@P+. Questo comando riattiva il pre-scan e' puo' essere riservato spazio per le variabili.

Il pre-scan puo' essere usato piu' volte in un programma ricordandosi di usarlo prima delle istruzioni SUB o SUBEND e senza essere seguito da altre istruzioni.

I seguenti esempi dimostrano come includere le istruzioni di pre-scan in un programma. L'esempio finale mostra l'uso piu' efficiente della caratteristica del pre-scan facendo uso dell'istruzione GOTO.

:sempi:

**Original program:**

```
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
130 .
140 .
150 .
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 DELAY=0
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 DATA 3
220 .
230 .
```

**With pre-scan control added:**

```
 10 DATA 3
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"OFOFOFOFOFOFOFOF")
125 !@P-
130 .
140 .
150 .
155 !@P+
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
180 DELAY=0
185 !@P-
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 .
220 .
230 .
```

Osservare che la prima istruzione DATA e' stata spostata all'inizio del programma in modo da includerla nel pre-scan. Inserendo le istruzioni 125, 155 e 185, il pre-scan e' disattivato, attivato e disattivato di nuovo. Cio' fa in modo che il programma inizi ad essere eseguito piu' rapidamente.

Con il GOTO aggiunto:
Il seguente esempio mostra il programma originale con un pre-scan ed un'istruzione GOTO.

```
 10 DATA 3
 20 GOTO 100::DELAY::CALL CHAR::CALL CLEAR::CALL HCHAR::CALL
    VCHAR::!@P-
100 CALL CLEAR
110 CALL CHAR(96,"FFFFFFFFFFFFFFFF")
120 CALL CHAR(42,"0F0F0F0F0F0F0F0F")
130 .
140 .
150 .
160 CALL HCHAR(12,17,42)
170 CALL VCHAR(14,17,96)
190 FOR DELAY=1 TO 500
200 NEXT DELAY
210 .
220 .
230 .
```

Nel programma seguente si mostra un piu' efficiente uso della opzione di pre-scan.

```
100 GOTO 180::X,Y,ALPHA,BETA,Z=DELTA::DIM B(10,10)
110 CALL KEY::CALL HCHAR::CALL CLEAR::CALL MYSUB
120 DATA 1,3,STRING
130 DEF F(X)=1-X*SIN(X)
140 .
150 .
160 .
170 !@P-
180 .
190 .
200 .
```

**Segni di commento**

In TI Extended BASIC e' possibile utilizzare il punto esclamativo quale segno di commento al programma oltre a REM.

# INDICE

# INDICE

# INDICE

# INDICE

# INDICE